TERADATA

Teradata Database

Database Administration

Release 12.0

B035-1093-067A

October 2007

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS-IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT WILL TERADATA CORPORATION BE LIABLE FOR ANY INDIRECT, DIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS OR LOST SAVINGS, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The information contained in this document 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.

Information contained in this document may contain technical inaccuracies or typographical errors. Information may be changed or updated without notice. Teradata Corporation may also make improvements or changes in the products or services described in this information at any time without notice.

To maintain the quality of our products and services, we would like your comments on the accuracy, clarity, organization, and value of this document. Please e-mail: teradata-books@lists.teradata.com

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed non-confidential. Teradata Corporation will have no obligation of any kind with respect to Feedback and will be free to use, reproduce, disclose, exhibit, display, transform, create derivative works of, and distribute the Feedback and derivative works thereof without limitation on a royalty-free basis. Further, Teradata Corporation will be free to use any ideas, concepts, know-how, or techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, or marketing products or services incorporating Feedback.

Preface

Purpose

This book gives an overall snapshot view on administrative concerns for creating and managing the objects that make up your Teradata Database. It also addresses topics such as the use of disk space, privileges, and introduces several tools for administering your system. Due to the breadth of topics covered, this book refers you to other manuals that discuss specific topics in greater detail.

Database Administration is not a tutorial. The Teradata Customer Education department provides courses on Teradata Database administration among its offerings. You can visit the Teradata Education Network by clicking its link on http://www.teradata.com/services-support to review or enroll in courses.

Audience

This book is intended for Database Administrators (DBAs) who are maintaining the Teradata Database configuration and data. It is also useful for operations control users.

Supported Software Release

This book supports Teradata® Database 12.0.

Prerequisites

You should have some knowledge of the following:

- Relational database concepts
- Teradata SQL
- The Teradata Database configuration
- The Teradata consoles:
    - On an MPP platform, the Administrative Workstation (AWS)
    - The Database Window (DB Window, or DBW)
    - On Microsoft Windows, the Teradata MultiTool

You might find it helpful to review the following publications:

- *Introduction to Teradata Warehouse*
- *Database Design*
- *SQL Reference: Fundamentals*

The following publications are directly related to the subject matter in this book and you may find it helpful to have them available for reference:

- *SQL Reference: Data Definition Statements*
- *Utilities*
- *Security Administration*
- *Data Dictionary*
- *Performance Management*
- The suite of Teradata Tools and Utilities client documents

For a list of upper limits for system, database, and session values, see the appendix titled "Teradata System Limits" in *Database Design*.

Changes to This Book

This book includes the following changes to support the current release.

Date	Description
Teradata Database 12.0 October 2007	This book contains the following changes: • Updated information about space usage views and re-emphasized recommendation for using DBC.TableSize and DBC.DiskSpace instead of DBC.AllSpace. • Included example for an optional optimized DBC.TableSize definition which avoids the possible problem of SELECT statements that sometimes hang. • Updated and added the following DBQLogTbl fields: UtilityRowCount, RowCount, and RowCount2, and ExtraField5. • Updated and added the following DBQLStepTbl fields: TDWMEstTotalTime, EstProcTime, StatementType, and ExtraField1. • Added note that new DBSControl flag that disables and enables CPU normalization of data within PM/API calls requires adjustment of CPU-related exception thresholds for Teradata Dynamic Workload Manager. • Updated example SQL code that used the alias "AMP" to "AMPno." The term "AMP" is a reserved word and cannot be used as an expression alias name. • Added warning about clearing out DBC.Acctg table when also using account string expansion (ASE).

Date	Description
Teradata Database 12.0 September 2007	The new content described in this manual includes the following: • Improved Optimizer cost profiles better handle nulls. • Query Banding for defining identifiers on queries to track their source. • Scandisk is restartable with the restartscandisk.pl script. • Index Wizard supports Partitioned Primary Indexes. • MP-RAS and Linux Kerberos authentication is now available for Windows clients. • Custom password dictionary allows users to add or remove words from a list of acceptable words. • Unicode Data Dictionary includes many improvements. • Using open APIs, you can code application interfaces between your OS and Teradata Dynamic Workload Manager in order to manage workloads affected by OS factors. • Teradata now supports Java External Stored Procedures. • Archive data using the ARC utility while the system is non-quiescent, that is, still online. • AmpUsage and DBQL views are normalized for coexistence systems. Changes include adding best practices for creating an administrative user, removing all information regarding crashdumps, and removing comma from sample code for resetting AmpUsage accumulators and from sample code in DBQL chapter before the keyword ACCOUNT. Improvements to Teradata Dynamic Workload Manager include: • The ability to create global rules, throttles and Workload Definitions. • Delay jobs rather than abort them when the system reaches concurrency limit maximums. • Create rules to disallow queries that result in specific kinds of joins or prevent queries that may use up AWTs. • Use specific Performance Groups according to current system conditions.

Date	Description
V2R6.2 September 2006	The following systems were supported by V2R6.2: Microsoft Windows Server 2003, MP-RAS, and Linux SUSE. New features for this release include the following: • The restrictions on defining join indexes and triggers on the same table has been lifted. You can now define both join indexes and triggers on the same table. • You can define and dynamically change the default value for a column of a table. • The File System creates a write cache to maximize I/O operation with Write Ahead Logging. • 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. • Restrict logons from predefined IP addresses. • Password encryption and control are enhanced. Also, tdgss is now a globally distributed object. • Passwords now default to using SHA-256 encryption. • Set session account to any valid account as long as the session is one to which you are assigned in DBC.Accounts. • Add, change, or remove multi-value compression at any time using the ALTER TABLE statement. Changes to this book include the following: • Added new object use count examples. • The REVALIDATE REFERENCES FOR statement no longer does validation for "soft" (no check option) referential integrity tables. • Before cancelling rollback, first submit a LIST ROLLBACK TABLES command and then cancel rollback on tables from that list. These steps must be done in the same Recovery Manager session. • DBQL updates include: • Statement type is now reported in DBQLObjTbl. • FastLoad and MultiLoad target tables are now reported. • You can specify PPI on non-compressed join indexes. The default directories used by Teradata on Linux have been changed to the following: • Configuration directory: /etc/opt/teradata/tdconfig • Temporary directory: /var/opt/teradata/tdtemp • Dump directory: /var/opt/teradata/tddump

Date	Description
V2R6.1 November 2005	The following systems were supported by V2R6.1: Microsoft Windows Server 2003, MP-RAS, and Linux SUSE. • Added warning about altering privileges for User DBC. • You can specify PPIs on global temporary tables and volatile tables. • Specify session level locks by setting the isolation level. See "Controlling Concurrency Through Isolation Levels" on page 218. • Control if all users, no users, or only user DBC logs on to Teradata Database. See "Changing Logon States and Restarting the System" on page 291. • The new awtmon utility reports AMP Worker Task usage and allows you to find busy AWTs. See "Finding Busy AMP Worker Tasks" on page 393. • Define a trigger in the dbschckrc resource file to trigger a script to run if system response time is above the "Expected Time Delay." • Set up UDFs and external stored procedures to do network I/O using an external security clause authorized under an OS user. See UDFs and external stored procedures in Chapter 2: "Building the Teradata Database." • Create your own user-defined types. See "User-Defined Types (UDTs)" on page 86. • Define additional password restrictions such as requiring mixed case, disallowing passwords to contain username, or requiring a minimum number of numeric, alpha, and/or special characters. • You cannot restore a database with join indexes or a table referenced by a join index. First drop the join indexes before restoring. • Updated book with changes to replication solutions. • Updates to CheckTable utility include the option to run CheckTable on an active production system, the option to report only problematic objects, the option to specify the number of tables checked in parallel in parallel mode, being able to get the warning messages in the summary report, and being able to get the summary information after an abort. • Added information about improving joins on DBQL tables by enabling the system to make ProcID and QueryID fields more unique. • Updated references of Sun ONE to Sun Java System Directory Server. • Added warning about altering the user "tdatuser" or the group "tdatudf." • Fixed the typo in the table "Granting Privileges" on page 172. The new privilege for Replication Services is REPLCONTROL. • Added restrictions about restoring or copying selected partitions. • Added information about user-defined types and methods. • A new external security clause allows you to assign user-defined functions and external stored procedures to a specific OS userid authorization.

Additional Information

Additional information that supports this product and the Teradata Database is available at the following Web sites.

In the following table, *mmyx* in the book product ID numbers B035-*xxxx-mmyx* represents the publication date of a manual, where *mm* is the month, *y* is the last digit of the year, and *x* is an internal publication code.

Type of Information	Description	Source
Overview of the release Information too late for the manuals	The Release Definition provides the following information: • Overview of all the products in the release • Information received too late to be included in the manuals • Operating systems and Teradata Database versions that are certified to work with each product • Version numbers of each product and the documentation for each product • Information about available training and support center	http://www.info.teradata.com/ Click **General Search.** In the Publication Product ID field, enter `1725` and click **Search** to bring up the following Release Definition: • *Base System Release Definition* B035-1725-067K
CD-ROM images	This site contains a link to a downloadable CD-ROM image of all customer documentation for this release. Customers are authorized to create CD-ROMs for their use from this image.	http://www.info.teradata.com/ Click **General Search.** In the Title or Keyword field, enter *`CD-ROM`*, and Click **Search.**
Ordering information for manuals	Use the Teradata Information Products Publishing Library site to order printed versions of manuals.	http://www.info.teradata.com/ Click **How to Order** under **Print & CD Publications.**
General information about Teradata	The Teradata home page provides links to numerous sources of information about Teradata. Links include: • Executive reports, case studies of customer experiences with Teradata, and thought leadership • Technical information, solutions, and expert advice • Press releases, mentions and media resources	Teradata.com

Type of Information	Description	Source
Additional information related to this product	Use the Teradata Information Products Publishing Library site to view or download the most recent versions of all manuals. Specific manuals that supply related or additional information to this manual are listed.	http://www.info.teradata.com/ Click **General Search**, and do one of the following: • In the Product Line field, select **Software - Teradata Database** for a list of all of the publications for this release. • In the **Publication Product ID** field, enter one of the following book numbers: • *Basic Teradata Query Reference* B035-2414-*067A* • *Data Dictionary* B035-1092-067A • *Database Design* B035-1094-067A • *Teradata Manager User Guide* B035-2428-*067A* • *Performance Management* B035-1097-067A • *Security Administration* B035-1100-067A • *SQL Reference: Data Definition Statements* B035-1144-067A • *Utilities* B035-1102-067A

References to Microsoft Windows and Linux

This book refers to “Microsoft Windows” and “Linux.” For Teradata Database 12.0, these references mean the following:

- “Windows” is Microsoft Windows Server 2003 32-bit and Microsoft Windows Server 2003 64-bit.
- “Linux” is SUSE Linux Enterprise Server 9 and SUSE Linux Enterprise Server 10.

Teradata plans to release Teradata Database support for SUSE Linux Enterprise Server 10 before the next major or minor release of the database. Therefore, information about this SUSE release is included in this document. The announcement regarding availability of SUSE Linux Enterprise Server 10 will be made after Teradata Database 12.0 GCA. Please check with your account representative regarding SUSE Linux Enterprise Server 10 availability in your location.

Table of Contents

Preface

SECTION 1 Introduction to Administering the Teradata Database

SECTION 2 Security and Authorization

SECTION 3 Availability

Chapter 8: Protecting Data

Chapter 10: Stopping and Restarting the System 283

SECTION 4 Housekeeping and Maintaining the Database

SECTION 5 Troubleshooting & Caveats

Chapter 14: Troubleshooting 383

SECTION 6 Appendixes

SECTION 1 Introduction to Administering the Teradata Database

CHAPTER 1 Overview

This chapter describes the initial objects on a newly installed Teradata Database system. This book assumes software installation and resource configuration have been completed. It also assumes that the proper installation scripts have been run and all initial set up has been performed.

What Makes Teradata Different?

Teradata saves you time and resources by freeing you from many day-to-day tasks such as planning when to defrag, reorg, or rebuild indexes. Teradata automatically handles many of the everyday data management tasks that would otherwise consume your time.

The File System Architecture

The Teradata File System is the mechanism that performs the storage, retrieval, and management of all rows in all tables stored in the database. Its architecture does not adhere to the page and buffer pool design established by the earliest database systems of years ago that are still used by other databases today[1].

A page-based database places all rows into fixed-size pages. Each row has a physical address that changes only when a reorg process is executed. The physical addresses are then used in other data structures (such as indexes) that constrain the ability of the database to move the row or page. The pages themselves have a fixed length; once a page is allocated to a table, it is fully allocated to the table regardless of whether it contains one row or many, or after most or all of the rows have been deleted. This results in poor space utilization and fragmentation. When a page is full, new rows that need to be inserted into that location must use overflow pages that result in performance degradation as the table matures over time.

However, Teradata has a very different design. It uses logical addressing rather than physical. By not tying a row to a physical location, the database gains the freedom to move the row at will without affecting in-flight work or any other data structures that reference the row. The rows are then stored in variable length "blocks," giving the database the freedom to grow and shrink the disk allocation on demand.

There are two levels of internal data structure to locate the row: the Master Index (MI) and the Cylinder Index (CI). With the MI pinned in memory, the user is always guaranteed a maximum of two I/Os to reach any row in the system; a guarantee that lasts regardless of how many updates, inserts, or deletes are performed on the table over an indefinite period of time. With the freedom to move rows, Teradata performs continuous allocation, garbage collection, and defragmentation. If numerous rows are inserted, blocks expand to contain the new data

[1] Todd Walter, "Who Manages Your Data - Your DBMS or DBA?," *TeradataReview*, 42-44, 1999.

until they reach the users’ specified maximum size, then automatically split to make room for more rows.

The following table summarizes the things automatically managed by the File System that make Teradata DBA-friendly.

Automatically Managed Entity	Description
Indexes	Indexes are managed as rows in internal tables. All of the automatic management of disk and location applies equally to indexes and rows in user tables. You do not need special structures, such as B-tress, to locate the rows. The way Teradata automatically manages indexes means you do not have to continually manage, fix, or rebuild indexes.
Disk Cache	The Teradata automatic disk cache adjusts dynamically to changes in workload and table access. The single disk cache requires no tuning on the part of the system administrator or database administrator (DBA) as the warehouse performs its daily cycles or as it evolves to meet the changing demands of the users and the business.
Space Management	Teradata does not set fixed table spaces such as other page-based databases. Teradata handles all disk space as a pool of available resources managed entirely by the file system. As a table, index, temp file, or log grows, the system allocates space for it from the pool. When it shrinks or is deleted, the space is returned automatically to the pool. To create a new table and populate it with data, submit a CREATE TABLE statement to define the columns and indexes and start inserting. Of course, there must be some control to keep a user or application from taking over all the disk space in the system. Each user or database has different quotas specifying the amount of disk space they may consume (how much perm, spool, and temp space). When users reach their quota, the system rejects further attempts to use more space. If the user does need more space, the DBA adjusts the quota (a one-second operation), and the user may immediately begin using more.
Overhead	Teradata performs all the automatic defragmentation and garbage collection under the control of thresholds. Until the system is run to the limit of its total disk capacity, almost no resources are expended. When required, the operations run concurrently in the background, using low-priority CPU and disk resources with minimum impact to the system workload.
Hash Partitioning	In a parallel system, the data must be partitioned in some way to allow for the parallel execution of work. In many cases, a DBA must determine and maintain this partitioning. A successful data warehouse, with its hundreds or even thousands of tables, can thus consume a large part of your DBA resources just to manage partitions. Teradata uses a hashing algorithm designed to distribute all data evenly among all partitions automatically. You do not have to define partitions when you define a table, and no maintenance is required as the data changes (for example, when new months, stores, or items are added, regions are defined, granularity of the data changes, or more history is added). You can, however, define partitions to be automatically managed by Teradata using partitioned primary indexes. For more information, see “Partitioned and Nonpartitioned Primary Indexes” in *Database Design*.

Automatically Managed Entity	Description
Reconfig	Upgrading a Teradata Database system to add new units of parallelism on additional hardware requires a *reconfig*. Prior to a reconfig, all new hardware components are brought up and connected with the warehouse fully active. Then the reconfig process is executed to re-partition the rows onto the new units of parallelism. While Teradata requires some amount of downtime for new hardware to be added, once reconfig is complete, there is never a need to postpone full usage of the data warehouse while table-by-table re-partitioning is performed. By bypassing the post-upgrade re-partitioning, you are free from having to do significant work on the affected table (possibly even reload), rework on the load jobs that manage the data in the table, and rework to the queries accessing the table. This step can amount to saving a six-month effort on a complex warehouse. Once the reconfig process is complete, the new hardware and units of parallelism are fully used immediately. The load utilities and the query optimizer automatically adjust to the new partitioning with no changes required. Reconfig is a scalable operation; large systems reconfig at the same speed as small ones.
Write cache	The system can automatically delay writing data in order to achieve a write cache. The system does Write Ahead Logging by writing to log records that reflect the updates made to the permanent data instead of writing the actual blocks at the time the transaction is processed. This significantly reduces the I/O write operations. By default, the system uses the write cache. However, if you do not wish to use a write cache, you can change the DisableWAL and DisableWALForDBs in the DBS Control Record. For more information, see “DBS Control (dbscontrol)” in *Utilities*.

Parallel From the Ground Up

In addition to the file system, the massively parallel architecture of Teradata allows it to optimally store and retrieve data. The file system, message subsystem, lock manager, and query optimizer all work together and all work in parallel. The parallel technology of Teradata allows you to do star joins and other join optimizations, efficient full table scans, quick processing with specially designed index types and so much more.

For more information, see “Born to Be Parallel: Architectural Origins of the Teradata Parallel Relational Database Management System” in *Database Design*.

Overview of Database Hierarchy

The file system and parallel architecture provide a great foundation for you to build your database upon. However, it is still crucial for you to carefully design and plan out your database. Before you begin, gather requirements and create a logical model of your database (see *Database Design* for more information).

After you have determined the best model for your data, you are ready to create users and databases. Users and databases are the logical repositories that make up a Teradata Database system. These users and databases contain other database objects such as tables, views, macros and so on.

When you first install Teradata Database on your server, it has only one user. This user is called User DBC and owns all other databases and users in the system. (See "Database DBC Contents" on page 36.)

User DBC also initially owns all the space in the entire system. As you create new databases and users, you subtract available permanent space from User DBC. 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. This is why User DBC is the owner of all subsequent owners in the whole entire database.

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

For information on creating users and databases, see "Creating a Database or User" on page 45. For information on creating an administrative user to manage the database rather than user DBC, see "Managing the Database with an Administrative User" on page 125.

Creating Versus Owning Objects in the Hierarchy

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

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.

An owner of an object is any database or user above the object in the database hierarchy. The immediate owner of a new user or database is the database space in which the new user or database resides. 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. For details, see "Creating a Database or User" on page 45.

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, such as 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. (See the following section "Changing the Hierarchy with GIVE.")

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

IF ...	THEN ...
you execute a CREATE statement that creates an object (anywhere in the Teradata Database)	you are the creator of that object. You are not necessarily an owner, or the immediate owner, of that object.
an object is directly below you in the hierarchy	you are the immediate owner of that object.

IF ...	THEN ...
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)	• you are the creator of the object. • the other user or database is the immediate owner of the object.
• UserA owns UserB • UserB creates an object in the database of UserB	• 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 and UserC is a directory-based user mapped to database-based UserA, and UserC creates a database in UserD	UserA is recorded as the creator and UserD is recorded as the immediate owner of the new database.

For information on privileges associated with owning and creating objects in the database see, "Chapter 6 Controlling and Tracking Privileges" on page 159.

Changing the Hierarchy with GIVE

Transfer databases and users from one immediate owner to another using the GIVE statement. GIVE transfers a database or user permanent space to a recipient you specify. 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.

Transfer of ownership affects space ownership and implicit privileges.

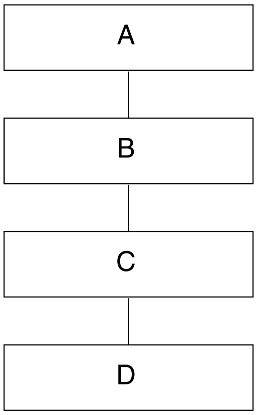
Rules affecting transfer of ownership include the following:

- You can transfer only databases and users with the GIVE statement. (GIVE does not operate at the table level.)
- 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 as needed.
- Changing ownership hierarchy impacts implicit privileges. That is, any database or user that no longer owns the database or user that is given to another immediate owner loses their implicit privileges on the database or user (and all objects below it in the hierarchy). The owners of the new immediate owner and the new immediate owner gain implicit privileges on the database or user (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.

Removing a Hierarchy Level

Assume that User A is an owner of Users B, C, and D. A is also the immediate owner of B, B is the immediate owner of C, and so on.



KY01A013

User A no longer needs User B, but wants to keep Users C and D. To remove User B from the hierarchy, User A must perform the following:

1 Transfer ownership of User C to User A: GIVE C TO A;

  The GIVE statement transfers both users C and D because the GIVE statement transfers the named user and all of its descendants in the hierarchy.

  Note that the explicit privileges of Users A, B, C, and D are not altered. The GIVE statement does not change explicit privileges that are defined in DBC.AccessRights.

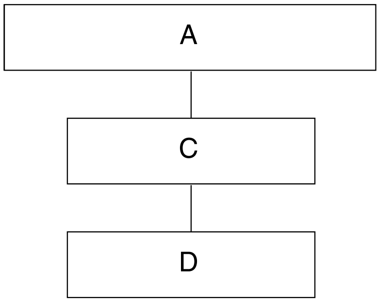
  However, User B does lose implicit privileges on Users C and D.

2 Delete all tables, views, macros, join indexes, triggers, stored procedures, functions, and journal tables from User B. All privileges on these objects are removed.

3 Drop User B. User A recovers the permanent space held by User B. It also modifies B to not use the journals.

  Note that you cannot drop the journal until all users of the journal are removed.

  Now the hierarchy is as shown in the following diagram.



KY01A014

Note: The explicit privileges for Users C and D remain intact. Although User B no longer exists, explicit privileges granted by User B are not automatically revoked. User A may use the REVOKE statement to change the explicit privileges of Users C and D on objects owned by User A or for which User A has the explicit privilege.

For the implications of how privileges are affected by GIVE, see:

- "Effect of the GIVE Statement on Privileges" on page 182.
- GIVE under "SQL Data Control Language Statement Syntax" in *SQL Reference: Data Definition Statements*.
- "Managing Data Access" in *Security Administration.*

Deleting and Dropping Objects in the Hierarchy

You cannot drop a database or user when any object exists in that space. Use DELETE to remove the objects first or re-assign them to another user by using the GIVE statement. Then you can submit a DROP statement to release the space.

DELETE DATABASE and DELETE USER

To delete objects in a database or user, use the DELETE DATABASE and DELETE USER statements. However, you must have the DELETE DATABASE and DELETE USER privileges. You must also have the DROP DATABASE or DROP USER privilege on a referenced database or user to delete objects from it.

After you delete objects from a database or user, the database or user remains in the Teradata Database as a named object. All space used by the deleted objects becomes available as spool space until it is reused as perm space. To completely remove an empty database or user, use DROP.

Note: Join indexes and hash indexes are not dropped by DELETE DATABASE or DELETE USER. Use DROP JOIN INDEX to drop a join index or DROP HASH INDEX statements to drop a hash index.

Note also that you cannot execute a DELETE DATABASE or DELETE USER statement if a table within the database or user references a journal table, join index, or hash index (according to the join index rule in regards to DELETE and DROP).

DROP DATABASE and DROP USER

To drop an empty database or user, use the DROP DATABASE and DROP USER statements.

The database or user that you are dropping cannot own other databases or users, or objects such as tables, join indexes, hash indexes, journals, macros, stored procedures, views, functions, or triggers.

For more detailed information, see "Dropping an Empty Database or User" on page 45 and "DROP DATABASE" and "DROP USER" in *SQL Reference: Data Definition Statements*.

The Initial Teradata Database

After the Teradata Database software is installed, it contains system user DBC. You can use the Database Initialization Program (DIP) utility to create special system users and databases although they probably have already been created during installation.

Database DBC Contents

Initially, the special system user DBC owns all space in the Teradata Database. Some space is assigned from DBC to system users and databases.

DBC is the default database associated with user DBC. 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
- SysAdmin user
- SystemFE user for field engineers
- The system Transient Journal (TJ)[2]
- The suite of special system query logging tables used to store query logging rules and data when you enable the Database Query Log (DBQL) feature. See "The DBQL Components" on page 345.
- The system catalog tables of the Data Dictionary and the suite of user-accessible views defined on those tables (for details, see *Data Dictionary*).

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

Teradata recommends that you create a special administrative user (different from the SysAdmin user) and that User DBC should grant privileges to this administrative user. The administrative user can manage most of the space available and become the owner of all administrator-defined application databases and users. See "Managing the Database with an Administrative User" on page 125 for more information.

Reserving Space for DBC

Teradata strongly recommends that you give sufficient space to 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 to 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.

[2] The TJ 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 your data from transaction aborts or to complete transactions.

An alternative is to create a database under DBC and allocate permanent space to be unused in order to hold the reserved spool space. (Assigning permanent space to DBC to be permanently unused is reserving this physical space for spool usage. See "Creating a Spool Reserve Database" on page 97.) The advantage of creating a database to hold spool space means that space will always be set aside and available for spool usage and will not be mixed with the space allocated for DBC.

DBC will use the space allocated to it for system, Transient Journal, and log processing. Reserving a specific amount of space from your total space in DBC helps ensure the entire system will run smoothly and properly because if DBC runs out of spool space, you may run into some problems with performance, unsuccessful queries, or user logon.

Note: Because systems vary significantly from site to site, analyze your own system to determine how much spool space to reserve for DBC. Some sites may require more while others require less. You should monitor spool usage and make adjustments later as needed.

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 to also regularly clean up old tables by deleting rows from them or archiving them instead if you need the data for later.

For further discussion on spool space, see "Spool Space and Capacity Planning" on page 93.

Other System Users

The DIP utility allows you to run one or more of the executable files that contain the SQL scripts that create system users, databases, and administrative tools. For example, executing the DIPCRASH script creates the Crashdumps user and its associated database. (For more information on the DIP utility, see *Utilities*.) The following section describes the different system users created by the DIP scripts.

SysAdmin Contents

User SysAdmin is created for internal use by Teradata system administration functions and contains several administrative views and macros, as well as a restart table for network-based FastLoad jobs. For information on the FastLoad client utility, see *Teradata FastLoad Reference*.

To protect these administrative objects and jobs, only a database or system administrator and Teradata personnel should log on as user SysAdmin.

Because unused permanent space that is not allocated to an object is available for spool or temporary space, SysAdmin is created with a small amount of permanent space for table storage.

SystemFE Contents

User SystemFE is created for internal use by Teradata system field engineers.

SystemFE is a created with a small amount of permanent space for tables. The following table lists the contents of SystemFE.

IF...	THEN SystemFE contains ...	For further information, see ...
the DIPSYSFE script has been executed	• macros that generate diagnostic reports for Teradata personnel logged on as this user. • Special privileges associated with user SystemFE are needed to execute the macros.	• *SystemFE Macros*. • *Utilities* for information on DIP scripts.
you enabled the Target Level Emulation (TLE) feature via the DBS Control Globally Distributed Object (DBSCONTROLGDO)	• special tables and macros used by Teradata Support Center personnel to perform Target Level Emulation (TLE) functions. TLE enables Teradata personnel to run your SQL applications on a configuration that emulates your Teradata Database. It is a useful way to test, diagnose, and debug your queries.	• "Target Level Emulation" in *SQL Reference: Statement and Transaction Processing*. • "Target Level Emulation" in the chapter titled "DBS Control Utility" in *Utilities*. • "Target Level Emulation (TLE)" on page 304.

To properly use SystemFE macros, DBC may need to grant SystemFE SELECT WITH GRANT OPTION on the following tables:

- DBC.ErrorMsgs
- DBC.Software_Event_Log
- DBC.Tables
- DBC.TableSize

Miscellaneous Databases/Users

The following table lists the users and databases created during the Teradata Database installation process at System Initialization (Sysinit) or database initialization time.

System Database/User	Description
ALL	Internal username; used by the database system software. Defined with no PERM space.
CRASHDUMPS[a]	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 PDE software. You may need to allocate more perm space, based on the size of your configuration, to accommodate at least three dumps.
DEFAULT	Internal username; used by the database system software. Defined with no PERM space.
PUBLIC	Internal username; used by the database system software. Defined with no PERM space.
Sys_Calendar[b]	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. For more information, see "CALENDAR System View" in *Data Dictionary*.
TDPUSER	Internal username; used by the Teradata Database to support two-phase commit (2PC) transaction protocol. Defined with no PERM space. For more information, see "2-Phase Commit (2PC) Protocol" on page 222 or *Teradata Director Program Reference*.

a. Defined to the Data Dictionary when the DIPCRASH script is run during the DIP phase of installation, migration, or upgrade.

b. Defined to the Data Dictionary when the DIPCAL script is run during the DIP phase of installation, migration, or upgrade.

DBC owns all other databases and users including the ones created by the DIP script during installation. Rather than use DBC to manage the system and risk accidentally changing the system users, you should create an administrative user to manage the system. (For more information, see "Managing the Database with an Administrative User" on page 125.) User DBC not only owns all other users and databases in the system but also has access to everything in the system because it has privileges to everything as well.

Database Privileges

A privilege[3] is the right to access or manipulate another object within Teradata Database. Privileges control user activities such as creating, executing, viewing, deleting, or tracking objects. Privileges also include the ability to grant privileges to other users in the database.

The following section introduces privileges; for more in-depth information, see Chapter 6: "Controlling and Tracking Privileges."

Introduction to Privileges

After you have reserved spool for user DBC and are ready to create databases and users to populate your system, you must decide what privileges to give to which users and databases.

Privileges are either explicit or implicit. Explicit and implicit privileges are usually in the context of a specific combination of:

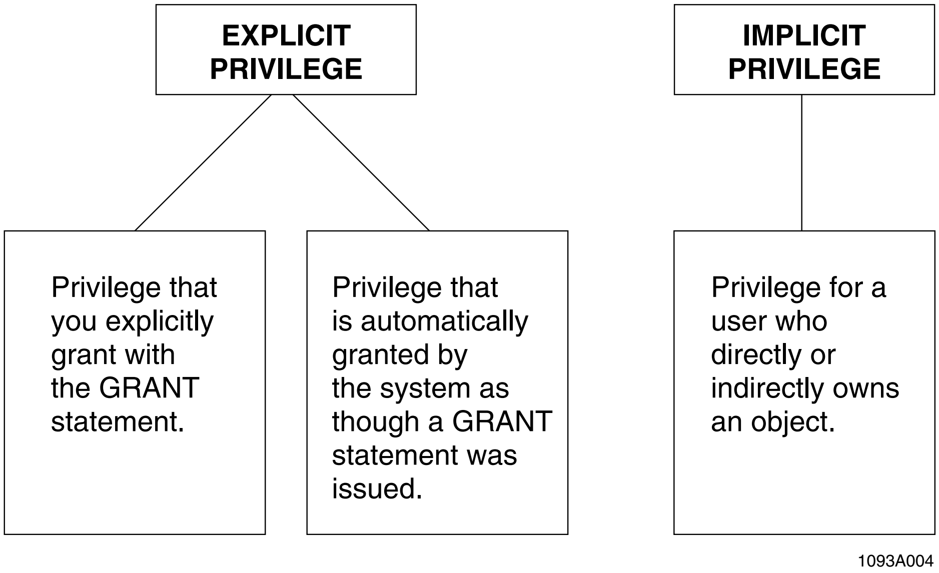
```
privilege ON object BY user/database/role/public [with or without GRANT
OPTION]
```

Some restrictions include:

- For some privileges, *object* is not applicable.
- For granting roles, *privilege* is not specified and is with or without DBADMIN option.

Explicit and Implicit Privileges

There are two kinds of privileges.


EXPLICIT PRIVILEGE
IMPLICIT PRIVILEGE
Privilege that you explicitly grant with the GRANT statement.
Privilege that is automatically granted by the system as though a GRANT statement was issued.
Privilege for a user who directly or indirectly owns an object.
1093A004

[3] Sometimes referred to as rights, access rights, or permissions. This book uses these terms interchangeably.

Explicit privileges:

- Are stored one row per user per right on a database object in the DBC.AccessRights table. Teradata Database uses this table to verify privileges for a user, database, or role.
- Are directly granted to a user or database on specific objects by another user (use of GRANT statement), or automatically granted to a user or database on objects it creates (use of CREATE statement).
- Can explicitly be granted to a role. A role is a collection of privileges on database objects. A user who is a member of a role can access all the objects on which the role has privileges. One row per role per right is stored in the DBC.AccessRights table.
- Can explicitly be granted to PUBLIC. Privileges granted to PUBLIC include every valid user of the system and all future users of the system. For further discussion of PUBLIC, see "PUBLIC Privileges" in *SQL Reference: Data Definition Statements*.

Implicit privileges are:

- Implied for an owner of an object.
- Implied from the ownership hierarchy defined in the DBC table.
- Sometimes referred to as ownership privileges.

The following table lists some general information about explicit and implicit privileges.

Explicit Privileges	Implicit Privilege
The creator of an object, who receives automatically granted explicit privileges, is the user who submits a CREATE statement or executes a macro containing a CREATE statement.	An owner of a user or database, who receives implicit privileges, is any user who owns the space from which that user or database was created. An owner can grant explicit privileges to other users, databases, roles, and PUBLIC and also grant the privilege WITH GRANT OPTION on the owned entity.
Teradata Database automatically assigns explicit privileges when an object is created. Explicit privileges are given through the GRANT statement or automatically granted to a user, database, role, or PUBLIC by another user with the appropriate privileges (implicit or explicit).	Implicit privileges are implied by ownership and no SQL statements need to be submitted.
When you submit a CREATE USER or CREATE DATABASE statement, the system automatically generates explicit privileges for both the creator and the created object. The system inserts the specific privileges into the DBC.AccessRights table. For a list of these privileges, see "Privileges Automatically Received" on page 163.	When you submit a CREATE USER or CREATE DATABASE statement, the owner of that newly created user or database automatically has privileges on it.

Explicit Privileges	Implicit Privilege
The GRANT statement adds new rows to the DBC.AccessRights or DBC.RoleGrants table. The REVOKE, DROP, and DROP/DELETE statements remove them.	Owners do not require rows in the AccessRights table to grant explicit privileges on owned objects.
By default, a created database or user cannot create databases or users, stored procedures, user-defined functions, roles, or profiles.	Implicit privileges cannot be refused or revoked. They cannot be taken away unless ownership is transferred or the owned entity is deleted or dropped.

For the special circumstance of granting rights to and revoking rights from PUBLIC, see "Granting the Ability to Grant Privileges" on page 173.

For more information on privileges, see Chapter 6: "Controlling and Tracking Privileges."

CHAPTER 2 Building the Teradata Database

This chapter describes the things that make up the Teradata Database, including:

- "Databases and Users"
- "Tables" (Permanent) and "Global Temporary and Volatile Tables"
- "Indexes"
  - "Primary Indexes (PI)" (Unique/Nonunique/Partitioned)
  - "Secondary Indexes (SI)" (Unique/Nonunique)
  - "Join Indexes (JI)"
  - "Hash Indexes (HI)"
  - "Collecting Optimizer Statistics on Indexes"
- "Triggers"
- "Macros"
- "Views"
- "Stored Procedures" (SQL and C/C++ or Java)
- "Permanent Journals"
- "User-Defined Functions (UDFs)", "User-Defined Types (UDTs)", and "User-Defined Methods (UDMs)"

Note: You can create the objects listed in this chapter using Teradata SQL. However, you can also create most of those objects using the Teradata Administrator. For more information see *Teradata Administrator User Guide*.

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. A user is the same as a database except it is an active repository that can log on to Teradata Database with a password and establish a session. (For more information, see "Creating Users" on page 127. For information on directory users, see "Directory Users" on page 130.)

Both may own objects such as tables, views, macros, procedures, and functions. Both users and databases may hold privileges. However, only users may log on, establish a session with the Teradata Database, and submit requests.

A user performs actions; a database is passive. Users have passwords and startup strings; databases do not. Users can log on to the Teradata Database, establish sessions, and submit SQL statements; databases cannot.

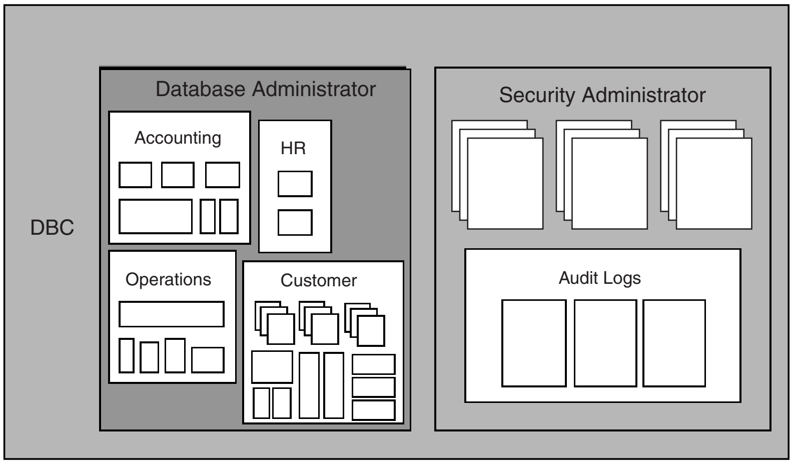
Creator privileges are associated only with a user because only a user can log on and submit a CREATE statement. Implicit privileges are associated with either a database or a user because each can hold an object and an object is owned by the named space in which it resides.

The maximum number of databases and users allowed on Teradata Database is $4.2x10^9$.

For more details on privileges, see "Creating Versus Owning Objects in the Hierarchy" on page 32 and "Overview of Access Privileges" on page 159.

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 a special administrative user that belongs to DBC (see "Managing the Database with an Administrative User" on page 125) and create your databases and users from the special administrative user.



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.

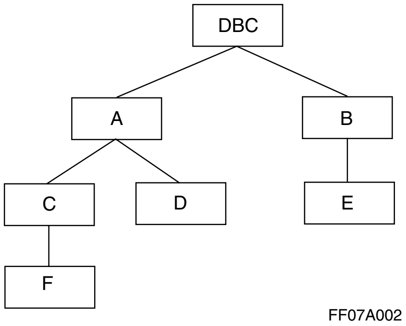
When creating additional databases or users, you must always allocate a permanent space limit. (You define space limits at the database or user level, not at the table level.) The MODIFY DATABASE/USER statement also allows you to specify the maximum limit of temporary, spool, and permanent space allocated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

Every database and user has an upper limit of temporary, spool, and permanent space. For more information, see Chapter 3: "Space Considerations."

Creating a Database or User

Users other than DBC (or a site administrative user with privileges on ALL objects in the database) 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 "Granting Create Privileges to a New User" on page 129.

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 (see "Increasing Space by Giving Ownership" on page 107).

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 and explicit privileges automatically granted to a creator or to a created database or user, see "Overview of Access Privileges" on page 159.)

Dropping an Empty Database or User

Use the DROP DATABASE/DROP USER statement only if the user or database to be dropped is empty. You must first delete all objects it contains, including data tables, views, global temporary tables, macros, stored procedures, functions, triggers, or indexes before you can drop that database or user.

Also, if a journal table exists, first be sure that no data table references it and then remove it with the DROP DEFAULT JOURNAL TABLE option of the MODIFY DATABASE/USER statement. For information on the SQL syntax, see *SQL Reference: Data Definition Statements*.

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 reorgs) do not apply to Teradata Database. For more information on automatic system maintenance, see "The File System Architecture" on page 29.

Tables

This section discusses creating and maintaining data tables for Teradata Database applications. You can load data to the tables with data using Teradata load utilities. For more discussion on those utilities, see "Client-based Utilities" on page 449.

Creating Tables

There are many things you can define about a table using the CREATE TABLE statement.

Use this statement ...	To ...
CREATE TABLE	create a new permanent table to have unique or duplicate rows, plus: • A name and data type for each of the columns • A primary index (PI) that can be defined as: • Single-column or multi-column • Nonpartitioned (NPPI) or Partitioned (PPI) • Unique or nonunique (UPI or NUPI, UPPI or NUPPI) • A single column defined with the IDENTITY attribute (to achieve row uniqueness during loading using system-generated values). Use the IDENTITY attribute on a column (if not the PI), to obtain a system-generated unique number without the overhead of a uniqueness constraint. Optionally, you can use the CREATE TABLE statement to define: • Fallback protection • Permanent journaling • Data block size, minimum data block size, and maximum data block size • Percent of each disk cylinder to leave free during data load • Referential integrity constraints (with or without checking) on column data • Secondary Indexes (SIs). See "CREATE TABLE (Index Definition Clause)" in *SQL Reference: Data Definition Statements*. • Compression of up to 255 distinct values per column, with a column limit determined only by the row length (because compressed values are added to the table header rows). • Queue tables. Queue tables are useful for queue-oriented applications such as event processing and asynchronous data loading for further processing. Queue tables are similar to ordinary base tables but behave like an asynchronous first-in-first-out (FIFO) queue. For syntax, considerations for, and examples on queue tables, see "CREATE TABLE (Queue Table Form)" in *SQL Reference: Data Definition Statements*.

Use this statement ...	To ...
CREATE *temporary_type* TABLE	create a temporary table where *temporary_type* is one of the following: • GLOBAL TEMPORARY (see "Global Temporary Tables" on page 55) • VOLATILE (see "Volatile Tables" on page 56).
RENAME TABLE	change the name of: • a re-created table to the original table name (after you drop the original) • an existing table (being careful about references to the original table name).

Making Changes to Tables

After creating a table, you can always go back and make changes to it. Use the ALTER TABLE statement to:

- Change certain PI or partitioning parameters
- Revalidate the PI (commonly used after an ARC RESTORE, especially when referential integrity or partitioning is defined on the table. For more information, see "Revalidating Primary Indexes" on page 59.)
- Change one or more of the following options:
  - Referential or other constraints at the table level
  - Whether or not to check a referential constraint
  - Fallback protection
  - Permanent journaling
  - Datablock size
  - Cylinder freespace percent
  - Primary or foreign key
- Drop inconsistent references (commonly used after an ARC RESTORE if DROP FOREIGN KEY does not work).
- Add or drop one or more columns.
- Add a column and specify one value or a list of values to compress.
- Change one or more column definitions (being careful about external references to the original column name), including:
  - Name
  - Data type
  - Data storage
  - Referential or other constraints at the column level.
  - Compression (add compression, change, or remove currently existing compression)
- On a global temporary table, change the option for:
  - Logging transactions
  - Retaining rows on transaction commit.

For more information, see "ALTER TABLE" in *SQL Reference: Data Definition Statements*.

Copying Tables

Use the AS option of the CREATE TABLE statement to copy some or all of an existing table. Several choice combinations are possible as described in the following table.

<table>
<tr><th>Use this statement ...</th><th>To copy an existing table as ...</th></tr>
<tr><td><i>CREATE TABLE AS</i> [<i>tablename/ query_expression</i>] ... <i>WITH [NO] DATA</i></td><td>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:
<table>
<tr><th>IF you want the table to inherit ...</th><th>THEN specify...</th></tr>
<tr><td>all of the column definitions plus the contents</td><td><i>tablename</i> WITH DATA and (for a volatile table) ON COMMIT PRESERVE ROWS.</td></tr>
<tr><td>all of the column definitions but none of the contents</td><td><i>tablename</i> WITH NO DATA.</td></tr>
<tr><td>a subset of the column definitions plus the contents</td><td>(<i>query_expression</i>) WITH DATA and (for a volatile table) ON COMMIT PRESERVE ROWS.</td></tr>
<tr><td>a subset of the column definitions but none of the contents</td><td>(<i>query_expression</i>) WITH NO DATA.</td></tr>
</table></td></tr>
<tr><td><i>CREATE GLOBAL TEMPORARY TABLE AS</i> [<i>tablename/ query_expression</i>] <i>...WITH NO DATA</i></td><td>a global temporary table. Use WITH NO DATA, because global tables are not populated until they are materialized by being referenced in a query.
<table>
<tr><th>IF you want the table to inherit ...</th><th>THEN specify...</th></tr>
<tr><td>all of the column definitions</td><td><i>tablename</i> WITH NO DATA.</td></tr>
<tr><td>a subset of the column definitions</td><td>(<i>query_expression</i>) and WITH NO DATA.</td></tr>
</table></td></tr>
</table>

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 *SQL Reference: Data Definition Statements*.

Dropping Tables

Use the following statements to drop tables.

TO remove a ...	USE this statement ...
permanent data table	DROP TABLE. 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. 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.
materialized global temporary table, before session end	DROP TEMPORARY TABLE. **Note:** Teradata 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 in order to:

- Change a default PI to a defined PI for a non-empty table.
- Change a NUPI to a UPI in a populated table when there is no USI.
- Redefine the partitioning on the PPI of a populated table. In some cases, you can still use ALTER TABLE; see “ALTER TABLE” in *SQL Reference: Data Definition Statements*.
- Change a data type attribute that affects existing data. For rules on changing data types, see “ALTER TABLE” in *SQL Reference: Data Definition Statements*.
- 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.

Use the SHOW TABLE statement 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.

However, the data type of every column in the new table must match the data type of the corresponding fields of the existing 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.allrights
   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.

3. If any data types 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 ;
   ```

4. Use DROP JOIN INDEX and DROP HASH INDEX to remove any join and hash indexes defined on the old table. Use SHOW JOIN/HASH INDEX *index_name* to modify, if needed, and then save the definition of each 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/INSERT … SELECT" in *SQL Reference: Data Manipulation Statements*.

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.

<table>
<tr><th>Issue</th><th>Considerations</th></tr>
<tr><td>Compression</td><td>Compression can reduce storage costs and enhance system performance. Use the COMPRESS phrase to compress up to 255 specific values or NULLs to zero space. (Nullable columns with null values are automatically compressed, even when the "COMPRESS" is not specified.)
Use ALTER TABLE to immediately add, change, or delete compression on a loaded or empty table.
You cannot compress the following:
• Component of the PI, whether partitioned or not
• Volatile table columns
• Identity column
• Derived table columns
• Spool table columns
• Referenced PK columns or referencing FK columns
For information on how to use compression, see “Value Compression” in <i>Database Design</i>.</td></tr>
<tr><td>Default value of a column</td><td>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.
<b>Note:</b> You cannot assign a 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>SQL Reference: Functions and Operators</i>. For more information on DEFAULT values and NULL substitutions returned to the host, see <i>SQL Reference: Data Types and Literals.</i></td></tr>
<tr><td>Default versus NULL value for aggregation results</td><td>During aggregation, the unspecified value represented by a:
• NULL value is ignored in the calculation.
• Default value is included in the calculation.
For example, assume you want the average salary of employees in Department 300 as follows:
<code>SELECT DeptNo, AVG(Salary)</code>
<code>FROM Employee</code>
<code>GROUP BY DeptNo</code>
<code>WHERE DeptNo = 300 ;</code>
If a salary is not known, the result differs depending on how the column is defined:
<table>
<tr><th>IF an unspecified value in the Salary column is defined with ...</th><th>THEN the result ...</th></tr>
<tr><td>a DEFAULT value</td><td>includes the default value as if it were the actual value. This may be far from the average result if all values were known.</td></tr>
<tr><td>NULL</td><td>is the average of the salaries that are known. NULL is ignored. This may be closer to the actual average.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Issue</th><th>Considerations</th></tr>
<tr><td>Default versus NULL value for aggregation results (continued)</td><td>You can use these tools to further control how unknowns are handled during aggregation:
<table>
<tr><th>IF you want to ...</th><th>THEN use the ...</th></tr>
<tr><td>exclude substitutions by making NULL the default value for an unknown</td><td>NULLIF function definition; for example:
NULLIF(SALARY,defaultvalue)
In this case, the <i>defaultvalue</i> is ignored in computing the average.</td></tr>
<tr><td>change an unknown represented by a NULL into the default value</td><td>COALESCE function definition. For example:
COALESCE(Salary,defaultvalue)
where <i>defaultvalue</i> is the value to be used in computing the average.</td></tr>
</table></td></tr>
<tr><td>Defining consistent data types and column names</td><td>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:
<table>
<tr><th>IF you are ...</th><th>THEN ...</th></tr>
<tr><td>using tables to store large data objects</td><td>create a table with large object (LOB) type columns:
• Character Large Object (CLOB)- A CLOB column can store character data, such as simple text, HTML, or XML documents.
• Binary Large Object (BLOB)- A binary large object (BLOB) column can store binary objects, such as graphics, video clips, files, and documents.
For more information on CLOBs and BLOBs, see <i>SQL Reference: Data Types and Literals.</i></td></tr>
<tr><td>joining the columns of one table with the columns of another table</td><td>ensure the following:
• 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, lastname firstinitial, such as "Smith H").</td></tr>
<tr><td>changing data in one table</td><td>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:
• A macro, a stored procedure, a trigger, or an application program to update all the tables affected by a change.
• Referential integrity constraints. (For details, see “Using Referential Integrity” on page 240.)</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Issue</th><th>Considerations</th></tr>
<tr><td>IDENTITY column</td><td>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. For more information, see “Using an Identity Column” on page 68.</td></tr>
<tr><td>Joins involving NULLs</td><td>
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:
<pre>
SELECT Name, Loc
FROM Employee, Department
WHERE Employee.DeptNo = Department.DeptNo;
</pre>
Undesired results may occur:
<table>
<tr><th>IF the ...</th><th>THEN the ...</th></tr>
<tr><td>row for new hire Smith in the Employee table contains a NULL in DeptNo, and the row for Administration in the Department table also contains a NULL in DeptNo:
<pre>
EmployeeTable   DepartmentTable
Name   DeptNo   Dept DeptNo
------ -----   ----- ------
Smith NULL   ADM   NULL
</pre></td><td>result contains:
• No information for Smith
• No information for any employee in Administration.</td></tr>
<tr><td>DeptNo column for Smith, Administration, and Engineering contain the same value (for example, NONE)
<pre>
EmployeeTable DepartmentTable
Name DeptNo Dept DeptNo
----- -----   ----- -------
Smith NONE   ADM   NONE
              ENG   NONE
</pre></td><td>WHERE condition for the query is satisfied. This links Smith with both Administration and Engineering, even though Smith works for only one department (which might not be either Administration or Engineering):
<pre>
Name   Loc
----- -------
Smith ENG
Smith ADM
</pre></td></tr>
</table>
To prevent an employee row with an unknown department number from matching an unrelated department row that also has an unknown department number, use a different default value to represent unknowns in the DeptNo column, as follows:
1 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.
2 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.
Note: 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.
For further discussion on the implications of using NULL and for information on how to design efficient joins, see “Joins and Hash Indexes” in *Database Design*.
</td></tr>
</table>

<table>
<tr><th>Issue</th><th>Considerations</th></tr>
<tr><td>NULL and the GROUP BY clause</td><td>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:
SELECT DeptNo, AVG(Salary)
FROM Employee
GROUP BY DeptNo;
The results can differ, as follows:<table>
<tr><th>IF the DeptNo column is ...</th><th>THEN the result ...</th></tr>
<tr><td>allowed to be NULL, and two employees have not yet been assigned a department number</td><td>lists the computed average for those two employees under a NULL department number. This might be confusing.</td></tr>
<tr><td>defined with DEFAULT and the specified constant is meaningful (such as Unknown)</td><td>is more valid.</td></tr>
</table>**Note:** 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.</td></tr>
<tr><td>Nullability of a column</td><td>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:<table>
<tr><th>IF no value is given for a column when a row is inserted, and the nullability of a column is...</th><th>THEN...</th></tr>
<tr><td>defined as NOT NULL</td><td>the INSERT statement returns an error.</td></tr>
<tr><td>not defined</td><td>a NULL is supplied automatically.</td></tr>
</table>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.</td></tr>
</table>

Global Temporary and Volatile Tables

This section provides the following information on:

- Global temporary tables
- Volatile tables
- Volatile tables versus derived tables
- Global tables versus volatile tables

Global Temporary Tables

Global temporary tables store a query result in a temporary table for use in subsequent queries. This improves system performance as follows:

- the system needs to re-execute the query fewer times.
- the user needs to create and drop fewer tables.

Use the CREATE GLOBAL TEMPORARY TABLE statement to create a global table. Global temporary tables are stored in the Data Dictionary, but their contents do not persist across sessions.

Also, their contents are not saved when a transaction completes, unless you explicitly specify otherwise. To save table rows after a transaction ends, specify ON COMMIT PRESERVE ROWS as the last keywords in the CREATE statement.

The following options are not available for global temporary tables:

- Referential constraints
- Permanent journaling
- Identity column

Space usage is charged to the temporary space of the login user.

You can materialize up to 2000 instances of global temporary tables in a single session, as long as your login user has adequate temporary space.

To obtain a list of all global temporary tables in the system[1], query the CommitOpt column of the DBC.Tables view. For example:

```
SELECT * FROM DBC.Tables WHERE CommitOpt IN (‘D’, ‘P’);
```

The CommitOpt column shows the value D or P for a global temporary table[2]. You can also query the TransLog column to determine if there is transaction logging for a global temporary table. For more information, see *Utilities*.

[1] To obtain a list of all global temporary tables you own, use the restricted (DBC.TableX) view.

[2] P is for ON COMMIT PRESERVE ROWS and D is ON COMMIT DELETE ROWS. N indicates the object is not a temporary table.

Materializing Global Temporary Tables

You can materialize a global temporary table locally by referencing it in an SQL data manipulation language (DML) statement. To do this, you must have the appropriate privilege on the base temporary table or on the containing database, as required by the statement that materializes the table.

Any number of different sessions can materialize the same table definition, but the contents change depending on the DML statements applied during the course of a session.

Privileges are not checked on the materialized instances of any global temporary tables because those tables exist only for the duration of the session in which they are materialized.

Space usage is charged to the temporary space of the login user.

For information on how to create global temporary tables, syntax definitions, and usage notes, see “CREATE TABLE (Temporary/Volatile Table Preservation Clause)” in *SQL Reference: Data Definition Statements*.

Volatile Tables

Volatile tables are useful for the following:

- creating tables that the system accesses and modifies frequently
- for when you know you do not need the data for later use
- automatically dropping the table at session end

Using volatile tables enables the Optimizer to do an index scan instead of a table scan.

The definition of a volatile table is stored in memory cache only for the duration of the current session. Space usage is charged to the spool space of the login user. If your SPOOL allocation is large enough, you can create up to 1000 volatile tables per session.

Note: Volatile tables do not survive a Teradata Database reset. Restart processing destroys both the contents and the definition of a volatile table.

The following are not available for volatile tables:

- Referential Constraints
- Check Constraints
- Permanent Journaling
- Compressed Column
- DEFAULT Clause
- TITLE Clause
- COLLECT STATISTICS
- Access rights checking (because volatile tables are private to the session in which they are created)
- Identity Column

For information on how to create volatile tables, syntax definitions, and usage notes, see “CREATE TABLE (Table Kind Clause)” in *SQL Reference: Data Definition Statements*.

Global Versus Volatile Tables

The following lists compare global tables and volatile tables.

Global Temporary Tables	Volatile Tables
• Materializes in the temporary space of the user • Base definitions are permanent in Data Dictionary tables • Definitions can be materialized by any user with the necessary DML privileges • Can be defined for COLLECT STATISTICS • Can survive a Teradata Database restart • Up to 2000 materialized tables per session • Materialized instance is local to a session	• Occupies space in the SPOOL allocation of the user • Definitions are kept in cache and go away at session end or during a Teradata Database restart • Private to the session in which they are created • Cannot be defined for COLLECT STATISTICS • Do not survive a Teradata Database reset • Up to 1000 tables per session • Local to a session
• If not dropped manually, instance is dropped automatically at session end • An instance an be dropped manually any time during a session with DROP TEMPORARY TABLE • Materialized contents are not shareable with other sessions • A very large number of global tables can prolong logoff, because materialized global tables are dropped automatically at session end.	• If not dropped manually, dropped automatically at session end • Can be dropped manually any time during a session with DROP TABLE • Contents are not sharable with other sessions • A very large number of volatile tables can prolong logoff, because volatile tables are dropped automatically at session end.

Indexes

An index is defined as a set of one or more table columns used by the SQL query Optimizer to evenly distribute data on the AMPs and efficiently retrieve rows. Teradata Database supports the following types of indexes:

- Primary
  - Unique and nonunique, nonpartitioned (PI and NUPI)
  - Unique and nonunique, partitioned (NPPI and NUPPI)
  - Identity Column used as a PI
- Unique and nonunique Secondary (USI and NUSI)
- Join (JI)
- Hash (HI)

Primary indexes serve to distribute and store the rows of a table, as a means of joining tables, and to enable direct-path access. Unique primary and unique secondary indexes also enforce row uniqueness.

Any combination of secondary, join, and hash indexes can be defined on one table. You can define up to a maximum of 32 indexes per table.

Keep in mind the following:

- Some load utilities cannot insert rows into a table that has been defined with an identity column, triggers, or secondary, join, or hash index. Refer to the specific utilities documentation for more information.
- An identity column cannot be defined as part of a join index, hash index, partitioned primary index, or value-ordered index.
- An ordered nonunique secondary index counts as two indexes against the 32 maximum.
- A Large Object (LOB) column cannot be a component of any index.

This section introduces each type of index and provides summaries of tools used for creating, changing, and dropping indexes.

Performance advantages can far outweigh the maintenance and resource costs of indexes. For a discussion of the pros and cons of using indexes, see *Performance Management* as well as "Selecting a Secondary Index" and "Join Index Benefits and Costs" in *Database Design*.

Primary Indexes (PI)

The following section introduces a few concepts about primary indexes.

Function

A PI performs vital functions and provides many benefits. The following table lists a few.

Function	Description
Enable efficient aggregations	The GROUP BY clause specified on the PI columns can result in a more efficient aggregation.
Enable efficient joins	An equality join constraint on the PI may enable a direct join to the target table. This eliminates the need to redistribute, spool, and sort rows before the join operation.
Provide access to rows more efficiently than block-at-a-time full-table scan	If you specify a query using a UPI, you access one row. If you specify a query using a NUPI, the system may access a few rows to find the desired row. With a PPI, even faster access is possible when you narrow the focus to only the qualifying partitions. **Note:** Other methods for fast retrieval include using USI values, a join index, a mix of index types, or a Cylinder Read full-table scan.
Distribute rows among the AMPs	A UPI always provides an even distribution.The column or columns defined for the NUPI affect how evenly rows are distributed across the disks. Because balanced distribution enhances AMP parallelism, choosing a good PI is usually critical to performance.

For information on how to choose and define indexes, see *Database Design*. For information on how to create indexes, see *SQL Reference: Data Definition Statements* and *Database Design*.

The basic rules of defining a PI for a new table include the following:

- The PI may be defined on one or many columns.
- The PI may be nonpartitioned (NPPI) or partitioned (PPI).
  - An NPPI may be unique (UPI) or nonunique (NUPI).
  - A PPI:
    - May be nonunique (NUPPI)
    - May be unique (UPPI) only if you include all the partitioning columns in the set of PI columns
    - Increases the row size by two bytes to indicate partitioning.

Because every table must have a PI, Teradata Database assigns one by default if you do not define one. For information on how the system chooses the default primary index based on whether or not you have defined a primary key or a uniqueness constraint, see “Primary Indexes” in *Database Design*.

A PI assigned by default probably does not provide the optimum row distribution of a table or the fastest access path to the rows. Instead, consider the following options:

- Choose the column or columns with values that best exploit the benefits of the PI. This ensures even distribution and direct data access.
- If a PI does not provide adequate results, consider using an identity column as your PI. The values of an identity column are generated by the system and are unique if properly defined and used. For more information, see *Database Design* or “CREATE TABLE (Column Definition Clause)” in *SQL Reference: Data Definition Statements*.
- Determine if partitioning of your primary index increases performance. Using PPIs means the system distributes rows to the AMPs based on the primary index hash value and then, on each AMP, store them according to an expression you define. For more information, see *Database Design* and *SQL Reference: Data Definition Statements*.

For NUPIs, try using the HASHBUCKET, HASHROW, HASHAMP functions to confirm that each index distributes the rows of its table evenly across the disks.

For information on PI use and selection, see “Selecting a Primary Index” in *Database Design.*

Revalidating Primary Indexes

The REVALIDATE PRIMARY INDEX option of the ALTER TABLE statement regenerates table headers and update indexes and internal numbering. Make it a practice to revalidate primary indexes for the following instances:

- The Archive/Recovery (ARC) utility completes a COPY/RESTORE operation after a migration.
- ARC completes an ARCHIVE/RESTORE operation for the following:
  - A Teradata Database upgrade
  - A Teradata Database with a different hashing algorithm (if partitioning expressions include the HASHROW or HASHBUCKET function)

- You change the RoundHalfwayMagUp field in the DBS Control record (if any partitioning expressions include decimal operations)
- Query responses or the CheckTable utility indicate incorrect partitioning

Submitted without options, REVALIDATE PRIMARY INDEX regenerates only the table headers. Define the *null_partition_handler* option of the REVALIDATE PRIMARY INDEX if, in addition to regenerating table headers, you also want to do the following:

- Update the partition numbers and row hash values.
- Move rows as necessary, based on the regenerated hash values and partition numbers.
- Update any SIs, JIs, and HIs defined for the table.

Caution: REVALIDATE does not correct rows that are out of order by their assigned row ID; an error occurs. If rows still have incorrect locations or internal partition numbers after revalidation, contact the Teradata Support Center.

The REVALIDATE process changes the table version number. You cannot execute certain ARC operations over a DDL statement that alters the table structure.

For more information on ARC operations, see the following:

- "Using ARC With Modified PIs or Partitioning" on page 253.
- Appendix B: "Import/Export Utilities."

Secondary Indexes (SI)

When using NUPIs, you can define a secondary index for a table to give the Optimizer another option for faster set selection. Although the Optimizer may still choose a full-table scan even if you define an SI, it generally chooses the SI if it is more useful for optimizing repetitive and standardized queries.

Secondary indexes are optional and are not related to row distribution. An SI is an index structure in the form of a subtable that is physically separate from the data rows of the base table. A row of an SI consists of an index value along with one or more identifiers of the base data rows having this index value.

For more information on SIs, see *Database Design*.

Creating a Secondary Index

You can define none, one, or many SIs on a data table, and any SI can be nonunique (NUSI) or unique (USI). Before you create a USI, consider the following:

- A USI is useful on the set of PI columns when the PI cannot be unique because it does not cannot contain all the partitioning columns.
- If a uniqueness violation occurs while attempting to create a USI, the entire transaction is rolled back.

- The rows of a USI subtable are hash distributed. This means USI subtable rows might reside on an AMP on which no data rows exist, and data rows might exist on an AMP on which no USI rows exist.

The following list describes basic rules for creating a SI:

- Create an SI on a new or existing table using the CREATE INDEX statement. SIs cannot be partitioned.
- Create USIs on the same set of columns as the PI if the PI is nonunique (NUPI or NUPPI). You can alter the table later to define the PI as unique if the PI is a PPI that includes all the partitioning columns or the PI is an NPPI.
- Define a value-ordered NUSI using the CREATE INDEX…ORDER BY VALUES statement.

Proving Secondary Indexes

During query processing, the Optimizer estimates comparative costs to determine whether the use of an SI offers savings or is plausible. If not, a full-table scan is used.

To achieve the best performance, it is good practice to test each SI before going into full production as follows:

- Emulate your production environment on a separate test system using TLE (see "Target Level Emulation (TLE)" on page 304).
- Generate a description of the processing sequence, which helps determine whether an existing SI should be retained or changed, using the:
  - SQL EXPLAIN modifier (see "SQL Data Manipulation Language Statement Syntax" in *SQL Reference: Data Manipulation Statements*).
  - Teradata VE (see "Teradata Visual Explain Utility" on page 306*)*.
  - Teradata Index Wizard, if you implement Query Capture Facility (see "Teradata Index Wizard" on page 306).

Join Indexes (JI)

A JI allows the system to get requested data from an index rather than having to access and join their underlying base tables. When a JI can be used to fully satisfy a query, you avoid the need to access multiple base tables, perform joins or aggregates, or redistribute the rows of very large tables, every time the query is run. You can create the following kinds of JIs:

- Single table or multitable
- Simple or aggregate

A JI table is hash-distributed, with possibly a different distribution of rows to the AMPs and row ordering on the AMPs from the base table rows.

Note: Value-ordered join indexes need to be dropped and rebuilt after a reconfig.

You can define one or more JIs on any table. You can also define one or more PPI or NUSIs on a table with a JI. If you want a NUSI on a join index to completely cover a query, use the ALL option if the join index is compressed or the NUSI contains case specific columns.

You can also specify a ROWID column in a JI definition. ROWID is a system-derived column that provides the internal row identifier associated with a row of a table. The principle use for ROWID values is to retrieve rows. This application is useful for enabling non-covering or partial-covering join indexes to join with base table columns.

Note: If a partial-covering JI is used to resolve a query, the join back to the base table is subtracted from the per-query join maximum.

There are two caveats for using JIs:

- You cannot restore a table or database with JIs. To restore a table or database with JIs, you must first drop the JI.
- If you want to define multiple join or hash indexes on a PPI table, be sure to plan for the increase in memory consumption. See “MaxParseTreeSegs” in *Performance Management*.

For more information, see “CREATE JOIN INDEX” in *SQL Reference: Data Definition Statements.*

Hash Indexes (HI)

HIs are similar to single-table join indexes and secondary indexes. They take columns most likely to be joined to another table and hash them to the same AMPs as the base table rows. This helps avoid having to redistribute the rows over the BYNET in order to make the join.

HIs can also provide another access path to the base table rows when the query uses columns not defined as part of the index. The system automatically updates HIs so the only task you must perform is to keep the statistics on the HI or base table columns up to date. (For more information, see “Refreshing Stale Collected Statistics” on page 300.)

The following table lists some things to keep in mind when defining a hash index.

IF ...	THEN ...
you define a trigger on a table	you cannot define a hash index on the same table. You must first drop the triggers.
you omit the ORDER BY clause in a CREATE HASH INDEX statement	the index rows default to the same rowhash value as the base table rows. Thus, the hash index rows are AMP-local to its base table and provide faster access. **Note:** For a PPI table, include ORDER BY (see “Secondary Indexes (SI)” on page 60).

IF ...	THEN ...
the columns of the order key are not unique	The Teradata Database system automatically compresses rows having the same order-key values into a single physical row having fixed and repeating parts, as follows: • The fixed part is made up of the columns that define the order key • The repeating part is made up of the remaining columns If all the columns do not fit into a single physical row, additional physical rows are allocated as necessary.
you want to define an HI (or multiple join indexes) on a PPI table	plan for the increase in memory consumption. See "MaxParseTreeSegs" in *Performance Management*.
do a reconfig on the system using the Reconfiguration utility	a value-ordered hash indexes need to be dropped and rebuilt.

For more information, see "Hash Indexes" in Database Design and "CREATE HASH INDEX" in *SQL Reference: Data Definition Statements.*

Costing and Validating Indexes

Trade-offs

When considering the choice of indexes and partitioning, carefully consider these trade-offs:

- Any query without conditions on defined indexes requires a full table scan.
- Secondary indexes, hash indexes, and join indexes incur overhead costs in space and maintenance performance during inserts, updates, and deletes.
- Partitioning a table can make your queries perform better if the partitioning column conditions allow the system to examine only a portion of the data instead of the whole table. However, there are some potential performance disadvantages as well. For more discussion, see "Partitioned and Nonpartitioned Primary Indexes" in *Database Design*.

Verifying Behavior

After you make your choice, verify the behavior for desired purposes such as these:

- If you chose a UPI to provide even distribution, verify that the rows are evenly distributed.
- If you chose a PPI to provide efficient access, validate that the PPI actually improves query performance.

As your data grows and changes, it is a good idea to collect statistics so that the Optimizer can more accurately determine which index performs the best.

For More Information

For guidelines and suggested tools, see the following:

- "Collecting Optimizer Statistics on Indexes" on page 64
- "COLLECT STATISTICS (Optimizer Form)" in *SQL Reference: Data Definition Statements*
- *SQL Reference: Statement and Transaction Processing*:
  - "Query Capture Facility"
  - "Database Foundations for the Teradata Index Wizard"

Dropping Indexes

If you have determined that a currently defined index is no longer the best choice, you may need to drop the index. The following table lists some statements you can use to drop indexes.

IF you want to remove a ...	THEN use ...	For more information, see ...
PI definition **Note:** A table must have a PI. You can only change a current PI, or remove the definition and let the Teradata Database system create a default PI. However, the default PI chosen by the system may not be an ideal choice. For details, see "Teradata Indexes: The Basics" in *Database Design.*	the following statements. To remove the definition completely, or make changes not allowed with ALTER TABLE, recreate the table with these statements: **1** CREATE TABLE **2** INSERT…SELECT **3** DROP TABLE **4** RENAME TABLE For other definition changes, you can use ALTER TABLE.	• To recreate the table, the procedure under "Recreating a Table" on page 49 • To alter the definition, ALTER TABLE in *SQL Reference: Data Definition Statements.*
SI	DROP INDEX	You can use SHOW JOIN INDEX, SHOW HASH INDEX, or HELP [JOIN/HASH] INDEX to review the PI or column definitions before changing or dropping an index.
JI	DROP JOIN INDEX	
HI	DROP HASH INDEX	

Collecting Optimizer Statistics on Indexes

Collecting statistics provides the Optimizer with information it needs to generate good query plans. The more accurate and up-to-date the statistics, the better the Optimizer can decide on and choose the fastest way to answer a query.

The COLLECT STATISTICS statement records statistical data about access to columns and the use of joins. The computed results are stored in the Data Dictionary for use during the optimizing phase of statement parsing. You can use Statistics Collection from the Production

Control menu of Teradata Manager to create, update, or view Optimizer statistics. (For more information, see *Teradata Manager User Guide*).

Note: Improved statistics collection now requires twice the amount of permanent and spool space for the Data Dictionary than in previous releases.

For more information on collected statistics, submit the HELP STATISTICS statement. For more information, see "HELP STATISTICS (Optimizer Form)" in *SQL Reference: Data Definition Statements*.

Usage

You should collect statistics using the Optimizer form on newly created data tables while they are still empty. An empty collection defines the columns, indexes, and synoptic data structure for loaded collections. You can easily collect statistics again once the table is populated for prototyping, and again when it is in production (see "The Importance of Re-Collecting Statistics" on page 66).

Use the Optimizer form of COLLECT STATISTICS to collect statistics on the following:

- Unique Index (primary or secondary)
  - Single or multiple column
  - Partitioned or nonpartitioned

    Collect partition statistics on all partitioned tables using system derived column PARTITION. This helps the optimizer do more accurate costing with partitioned tables which helps produce more optimal plans.
- Nonunique Index
  - Primary or secondary
  - Single or multiple column
  - Partitioned or nonpartitioned
  - With or without COMPRESS fields
- Non-indexed column or set of columns
  - Partitioned or nonpartitioned
  - With or without COMPRESS fields
- Join index (JI)
- Hash index (HI)
- Sample (system-selected percentage) of the rows of a data table or index, to detect data skew and dynamically increase the sample size when found.
  - The SAMPLE option is not supported for global temporary tables, JIs, or HIs.
  - The system does not store both sampled and defined statistics for the same index or column set. Once sampled statistics have been collected, re-collection is implicit on the same columns and indexes, and operates in the same mode. To change this, specify any keywords or options and name the columns or indexes.

Note: Copy statistics when copying a table with the COPY TABLE AS… WITH DATA AND STATISTICS statement. See "Copying Statistics From a Base to a Target Table" on page 48.

The Importance of Re-Collecting Statistics

In order for the Optimizer to make best use of indexes, set up a schedule to regularly collect statistics.

It is important for the Optimizer to have accurate demographics on the PI to determine whether row distribution is skewed (that is, if distribution across the AMPs is uneven).

Skewing might occur when the following are true for a table:

- The PI is not unique (a NUPI or NUPPI)
- More than 10 percent of the PI values are the same
- A UNIQUE column or column set, such as a USI or an identity column (See "Using an Identity Column" on page 68), are not defined or not used.

If data is skewed or the access path the Optimizer chooses is not optimal, modify the PI and re-collect statistics until desired results are indicated. You can use the "Query Capture Facility (QCF)" and "Teradata Visual Explain Utility" to analyze the differing plans. For a brief overview, see the descriptions of each tool under "Housekeeping Queries" on page 304.

To update demographics automatically, enter the COLLECT STATISTICS statement with only the tablename (omit column or index specifications):

```
COLLECT STATISTICS ON table_name ;
```

The parser then does as follows:

- Determines what columns or indexes of the table already have a statistics structure in the Data Dictionary.
- Re-collects statistics on those columns and indexes in the same node (SAMPLE or not) as previously collected.

Note: This statement form does not support the USING SAMPLE clause. If you need to re-collect statistics for a SAMPLE clause, submit another COLLECT STATISTICS statement with explicit references.

With fresh statistics, you can improve the optimization of query plans. For example, if you ask a query with a DATE range containing a date further than what is available, the Optimizer can extrapolate data from the statistics and provide a good estimate.

Statistics on Null and AllNulls

When you collect statistics, Teradata Database records two different counts of null values:

```
help stats table1 column (y,z);

 *** Help information returned. One row.
 *** Total elapsed time was 1 second.

               Date  05/12/20
               Time  10:55:40
     Number of Rows         8
    Number of Nulls         5
 Number of AllNulls         3
    Average AMP RPV         2
Number of Intervals         1
  Number of Uniques         5
            Numeric         N
            Sampled         0
    Sampled Percent         0
            Version         2
          Min Value         1,1
         Mode Value         5,5
     Mode Frequency         2
          Max Value         5,5
         Mode Value         1,1
     Mode Frequency         1
   Non-Modal Values         -2
     Non-Modal Rows         2
```

- Null is the count of the number of rows that have a null value in one of its fields. This includes rows with all its fields as null.
- All-Null is the count of only the rows that have *all* its fields as nulls.

For example, in the following table:

```
x          y
------    -------
1          a
?          b
?          ?
4          ?
?          ?
```

Null count is 4 and All-Null count is 2.

Use the HELP STATS statement to report the number of nulls after collecting statistics. By considering both Null and All-Null counts, the Optimizer can better estimate the number of unique values in the presence of nulls.

Populating Tables

Populate tables using a Teradata load utilities such as FastLoad, MultiLoad, BTEQ or TPump. The following factors affect which utility you should choose:

- If you have defined join or hash indexes
- If you have defined secondary indexes (For example, FastLoad cannot load tables defined with any secondary indexes. MultiLoad can load tables with NUSIs but not USIs. FastLoad cannot load tables with duplicate values in tables defined as MULTISET.)
- The size of your table
- Whether the you are loading a moderate or a large amount of data

For a comparison and some guidelines on which load utility to use, see "Client-based Utilities" on page 449.

Note: Teradata recommends that you run tests on an adequate amount (approximately 10 percent) of real data before choosing a utility.

The following sections discuss additional considerations when populating tables with data.

Loading Tables with Triggers

To load data into a trigger-enabled base table, you must disable all defined triggers before you can run the FastLoad or MultiLoad utility. To disable a table that has more than one trigger, you can use the statement:

```
ALTER TRIGGER tablename DISABLE ;
```

Supplying a table name instead of a trigger name disables all triggers for that table (see "ALTER TRIGGER" in *SQL Reference: Data Definition Statements*).

BTEQ and load utilities like TPump, which perform standard SQL inserts and updates, are supported for tables on which triggers are enabled.

Loading Tables with PPIs

You may use load utilities for a table with a PPI but with the following conditions:

- FastLoad does not support SIs of any kind (and tables with PPIs also often have USIs). However, you can load the tables first and create the SIs later.
- MultiLoad does not support USIs
- The MultiLoad IMPORT task does not support updates of:
  - The PI columns (whether PPI or NPPI)
  - The partitioning columns.
- For update/deletes, MultiLoad IMPORT task requires:
  - All values of the PI columns
  - All values of the partitioning columns.

Using an Identity Column

IDENTITY is an optional attribute of the CREATE TABLE statement and causes the system to generate a number for every row inserted into the data table on which it is defined. The identity column does not have to be the first column in the table or be defined as an index.

To make the identity column numbers unique, you must use the GENERATED ALWAYS and NO CYCLE options of the CREATE TABLE. You can then avoid load preprocessing by eliminating duplicates when loading tables and achieve uniqueness of PK values without incurring the overhead of a UNIQUE constraint.

When you use an identity column as the PI, you can accomplish the following:

- Achieve row uniqueness without the need for a composite index.
- Ensure row uniqueness during a merge of several tables that each have a UPI. The identity column PI is the UPI for the final result table.

When you create an identity column on a table, the system performs one of the following:

- Single INSERT statements made through multiple concurrent sessions (BTEQ IMPORTs into the same table)
- Multiple INSERT statements made through multiple concurrent sessions (TPump inserts)
- INSERT-SELECT statements
- Atomic UPSERTs (with a non-PI identity column)
- Single-row MERGE-INTO.

For more information on using an identity column, see *Database Design* or see "CREATE TABLE (Column Definition Clause)" in *SQL Reference: Data Definition Statements.*

Loading Identity Column Tables

You can do the following types of loads on tables with identity columns:

- Single-statement INSERT requests through multiple concurrent session; for example, BTEQ IMPORTs into the same table
- Multi-statement INSERT requests through multiple concurrent sessions; for example, TPump (ROBUST mode is recommended).

You can load tables with identity columns with FastLoad or MultiLoad using Teradata Parallel Transporter.

Note: When you SELECT on the DBC.IDCol table, Teradata Database implicitly uses an ACCESS lock instead of a READ lock that is normally used for all SELECT operations. You can explicitly force a READ lock through the LOCKING TABLE FOR READ request modifier to SELECT from DBC.TDCol. However, this may result in a deadlock or a hang when used inside a BT/ET statement.

Triggers

A trigger contains one or more stored SQL statements that are executed, or fired, when some other event, called a triggering event, occurs. Triggers must be associated with an event and cannot be executed independently.

A trigger is associated with a subject table, and is stored as a named database object. Triggers exist in enabled or disabled states; when disabled, triggers remain as inactive database objects.

Typically, triggers execute when an INSERT, UPDATE, or DELETE modifies one or more specified columns in the subject table. The stored statements then perform operations such as INSERT, UPDATE, or DELETE on indicated tables, which may include the subject table.

Types of Triggers

There are two kinds of triggers.

This type of trigger...	Fires once for each...
Row	row changed by the triggering statement.
Statement	triggering statement.

Trigger Action Time

You can specify when triggers fire in the CREATE TRIGGER statement.

WHEN you specify...	THEN the triggered action executes...
BEFORE	before the completion of the triggering event. **Note:** A BEFORE trigger cannot have data-changing statements as triggered action (triggered SQL statements).
AFTER	after completion of the triggering event.

Note: According to ANSI semantics, if the subject table, on which a trigger is defined, is referred anywhere inside a BEFORE trigger or inside an AFTER trigger, then those references would be able to see the rows before or after the operation on the subject table, respectively.

The Teradata Database processes and optimizes the triggered and triggering statements in parallel to maximize system performance.

Creating and Using Triggers

A trigger is defined by a CREATE TRIGGER statement. Use the ALTER TRIGGER statement to enable or disable a trigger without having to drop the trigger entirely. ALTER TRIGGER also allows you to change the creation timestamp used to define its order of execution. For more information, see “ALTER TRIGGER” and “CREATE TRIGGER REPLACE TRIGGER” in *SQL Reference: Data Definition Statements.*

Utilities such as FastLoad and MultiLoad cannot operate on tables defined with enabled triggers. However, you can bulk load these tables if you disable the triggers. If you disable triggers to use FastLoad and MultiLoad, remember that triggers sometimes define relationships between tables as an extension to Referential Integrity. Therefore, when disabling triggers to use FastLoad and MultiLoad, be careful that relationships maintained with triggers enabled are still maintained by the application using the load utilities.

Functions of a Trigger

You can use triggers to perform a variety of functions:

- Define a trigger on the subject table to maintain a user defined relationship between the subject table and other tables. This is an extension of the Relational Integrity concept.
- Use triggers for auditing. For example, you can define a trigger that causes INSERTs in a log record when an employee receives a raise higher than 10 percent.
- Use triggers to set thresholds. For example:
  - Set a threshold for inventory of each item by store.
  - Create a purchase order when the inventory drops below a specified level.
  - Change the price of an item if the daily sales do not meet expectations.

Restrictions on Using Triggers

Restrictions on triggers include the following:

- If your applications use triggers, you must disable triggers before running a load utility because triggers are not supported by most Teradata load utilities.
- You cannot combine row and statement operations within a single trigger definition.
- BEFORE statement triggers are not allowed.
- You cannot define triggers on a table that already has a hash index defined.
- A positioned (updatable cursor) UPDATE or DELETE is not allowed to fire a trigger. An attempt to do so generates an error.
- You cannot use a SET QUERY_BAND statement in a trigger.

For more information on triggers, see “CREATE TRIGGER REPLACE TRIGGER” in *SQL Reference: Data Definition Statements*.

Macros

Teradata Database macros are SQL statements or multi-statement requests that are stored as a database object under a unique name. You execute the stored statements by submitting the EXECUTE MACRO *macroname* statement. The privileges required on the underlying tables are verified for the immediate owner of the macro.

IF you want to ...	THEN use the statement ...
define and store a new macro	CREATE MACRO **Note:** A CREATE MACRO statement containing an EXECUTE MACRO statement succeeds even if the macro in the EXECUTE statement is not yet defined. Similarly, a CREATE MACRO statement that contains a DDL statement succeeds even if the DDL object is not yet defined. For security reasons, the CREATE AUTHORIZATION or REPLACE AUTHORIZATION statements are not allowed within a macro.
run the SQL statements within the macro	EXECUTE MACRO
modify the macro	REPLACE MACRO **Note:** A REPLACE MACRO statement containing an EXECUTE MACRO statement succeeds even if the macro in the EXECUTE statement is not yet defined. Similarly, a REPLACE MACRO statement that contains a DDL statement succeeds even if the DDL object is not yet defined.
remove a macro	DROP MACRO Be careful that applications do not reference the macro being dropped.

The advantage of using a macro is easy execution of frequently-used SQL operations. The statements within the macro can also be defined to do the following:

- Enforce data integrity rules
- Provide data security

For More Information

For more detailed information on macros, see the following:

- "Limiting Data Access with Macros" on page 198
- "Macro and Tactical Queries" in *Database Design*
- "CREATE MACRO REPLACE MACRO" in *SQL Reference: Data Definition Statements*

Views

A view is a virtual table created by defining a SELECT statement on one or more base tables or other views. When a user references a view, the response returns the rows selected according to the CREATE VIEW viewname AS … SELECT FROM … definition. Thus, you can think of a view as a dynamic window to its underlying tables.

Views can also select and return data from one or more columns of a combination of one or more base tables and other views. You can enforce access privileges by granting only certain users the privilege to use the view. Or you can use a WHERE clause to qualify particular rows and to perform functions such as SUM or AVERAGE.

Note: You can create hierarchies of views in which views are created on views. This can be useful, but be aware that deleting any of the lower-level views destroys dependencies of the higher-level views in the hierarchy.

There are several good reasons to use views:

- A simplified user perception of very large and complex tables
- Security, by omitting to select the sensitive columns of the underlying data table
- Well-defined, well-tested, high-performance access to data
- Logical data independence, which minimizes the need to modify your applications if you restructure base tables
- A column defined in a view can be derived and does not need to exist in the underlying base tables. For example, it is possible to display summed or averaged data in a column you name in the CREATE VIEW definition.

For more detailed information on views, see the following:

- “Limiting Data Access with Views” on page 195.
- “CREATE VIEW” in SQL Reference: Data Definition Statements for syntax information.
- “Rules for Updating Rows Using Views” and “Deleting Rows Using Views” in *SQL Reference: Data Manipulation Statements*.

Stored Procedures

A stored procedure is a set of compiled procedural statements and SQL statements. The procedural statements make it possible to write significant portions of complex applications purely in SQL.

Stored procedures also enable you to develop and execute certain types of SQL statements that cannot be performed interactively, such as the following:

- Cursor control and declaration statements
- Flow control statements and condition handlers
- Nested compound statements

- Dynamic SQL statements
- Special forms of SQL statements (for example, SELECT INTO)

The client application is relieved of having to perform many intermediate steps because users can create, compile, store, execute, replace, and recompile their procedures directly on the server instead of on the client. For statement syntax and complete usage details see *SQL Reference: Stored Procedures and Embedded SQL*.

Administrative Considerations

The named set of compiled SQL constructs is stored in permanent table space.

The ALTER PROCEDURE statement allows you to recompile stored procedures without the need for SHOW followed by REPLACE. This is useful for bulk recompilation of stored procedures during a Teradata Database upgrade or migration, or cross-platform copy/restore of individual stored procedures.

For information on the privileges required for stored procedures, see "Privileges That Must Be Explicitly Granted" on page 165.

Supporting Client Utilities

Stored procedure execution and DDL operations are supported by many Teradata client facilities, including those listed in the table below.

IF you write for ...	THEN use the ...
BTEQ	.COMPILE BTEQ command
CLIv2	CREATE PROCEDURE/REPLACE PROCEDURE SQL statements
JDBC	
ODBC	
Teradata SQL Assistant	

Note: PreProcessor 2 (PP2) supports stored procedure execution, but not creation.

For More Information

FOR more information on...	SEE...
how to create and use stored procedures	*SQL Reference: Stored Procedures and Embedded SQL*
which data definition statements you can define in stored procedures	Supported DDL Statements in Stored Procedures in *SQL Reference: Data Definition Statements*.
using stored procedures to control user access to data	"Limiting Data Access with Stored Procedures" on page 199.

External Stored Procedures

External stored procedures are written in C, C++, or Java. You install them on the database and then execute them like stored procedures. To install an external stored procedure, you must have the CREATE EXTERNAL PROCEDURE privilege on the database.

C/C++ External Stored Procedures

You can write stored procedures in C or C++. External stored procedures are called like any other stored procedure via the CALL statement. They can execute SQL indirectly by making a C library call to invoke an SQL stored procedure or execute SQL directly using CLIv2.

To allow external stored procedures to access resources, such as local or network files, design the stored procedure to run in secure mode. In secure mode the procedure is assigned to use a specific OS userid authorization. To accomplish this, first specify the EXTERNAL SECURITY DEFINER/INVOKER clause in the CREATE PROCEDURE statement. Then, assign an OS user using the CREATE AUTHORIZATION statement. Any OS user that is given authorization must belong to the "tdatudf" group.

Note: External stored procedures using CLIv2 to directly execute SQL statements can run in secure or protected mode, use an EXTERNAL NAME clause that specifies the CLI package, and specify either a CONTAINS SQL, READS SQL DATA, or MODIFIES SQL DATA clause.

External stored procedures not accessing any OS resources can run in non-protected mode without the external security clause. However, external stored procedures must run in protected mode whenever they perform operations that require the OS to keep track of activities (for example, open files, connect to network resources, use semaphores or events, and so on).

External stored procedures can also use UDTs as parameters and return values. For a brief overview on UDTs, see “User-Defined Types (UDTs)” on page 86.

For more information on programming and administration of external stored procedures, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.

For information on privileges required for external stored procedures, see “Privileges That Must Be Explicitly Granted” on page 165.

Java External Stored Procedures

You can write external stored procedures in Java and use JDBC to dynamically execute SQL within the same session. To use Java external stored procedures, first set up the Java archive (JAR) files which contain the Java classes. For information on programming and administrating JAR files, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.

Note: You can only use Java external stored procedures on 64-bit Windows and Linux and run them in either secured mode or protected mode. Java external stored procedures do not support GRAPHIC and VARGRAPHIC values or UDTs as parameters and return values. You need to specify either a CONTAINS SQL, READS SQL DATA, or MODIFIES SQL DATA clause for a Java external stored procedure executing SQL via JDBC.

To begin administrating JAR files, first register them with the database and crate object names for them. JAR files created for one external stored procedure on one database cannot be shared with other users and databases.

For example, user/database A cannot create an external stored procedure which references a JAR installed in user/database B. However, user/database A can be granted access to a Java external stored procedure which has been created in user/database B, by using the same privileges designed for C/C++ external stored procedures. Or, to simplify, you could install all JAR files and create all Java external stored procedures in the same database, and then grant access to these Java external stored procedures to all users who will need to execute them.

Once you create the Java external stored procedure, you can access it in the same manner as you would any other external stored procedure. Java external stored procedures can execute SQL code through the standard JDBC driver interface. Since the stored procedure is running on the database and is invoked from within a logged on session, the connection to the database is made through a default connection ("jdbc:default:connection").

Note: It is still possible for the Java program to create a separate connection to another database or the same database (through another session to be logged onto). When doing so, it is possible to create a deadlock that cannot be detected. It is also important to note that Java external stored procedures which use JDBC cannot be multithreaded.

The ALTER PROCEDURE statement allows the user to recompile or re-link the function and redistribute it and its associated JAR file through a compile option. In particular, use this statement to recompile created functions that have been moved to another platform, system or restored database.

Permanent Journals

The purpose of a permanent journal is to maintain a sequential history of all changes made to the rows of one or more tables. Permanent journals help protect user data when users commit, uncommit, or abort transactions. A permanent journal can capture a snapshot of rows before a change, after a change, or both.

You use permanent journaling to protect data. Unlike the transient journal, the contents of a permanent journal remain until you drop them. When you create a new journal table, you can use several options to control the type of information to be captured.

The following table describes the options.

Option	Description
Single Image	Captures/stores one copy of the data.
Dual Image	Captures/stores two separate copies of data: one copy on the primary AMP and one on the fallback AMP.
Before Image	Captures/stores row values before a change occurs.
After Image	Captures/stores row values after a change occurs.

Journaling Protection

Use permanent journal tables to protect against the following:

- Loss of data caused by a disk failure in a table that is not fallback or RAID protected
- Loss of data if two or more AMP vprocs fail in the same cluster. This would mean the loss of two disks in a rank per failed AMP vproc. (For information on AMP distribution in a cluster, see "AMP Clustering and Fallback" on page 231.)
- Incorrect operation of a batch or application program
- Disaster recovery of an entire system
- Loss of changes made after a data table is archived
- Loss of one copy of the journal table (with dual journaling).

Journal Location

Journal tables are allocated permanent space and reside within a database or user space. Each database or user can contain only one journal table.

You create permanent journals with a CREATE USER/DATABASE or MODIFY USER/DATABASE statement.

Data tables can write to a journal in the database or user that owns them, or to a journal located in another database or user.

A journal in a database or user is the default journal for data tables in the same database/user, but you can specify that a data table write to a different journal in the CREATE TABLE or ALTER TABLE statement.

Rollback with Before-Image Journals

Before images are used for ROLLBACK recovery. Once a before-image journal is created, a snapshot of an existing row is stored in the journal table before any data is modified.

In the event of a software failure, the before-image journal can roll back any unwanted changes in the data tables that write to that journal.

Permanent journals roll back all transactions from a table to a checkpoint. They may not be used to roll back specific transactions.

Rollforward with After-Image Journals

An after-image journal stores a snapshot of a row value after a change has been committed. Then if a hardware failure occurs, you can use the after-image journal to roll forward any changes made to the data tables that write to that journal since the last full system backup.

To protect against the loss of data in the event of a site disaster, many applications require that data archives be kept off-site at all times. Ideally, users dump the database to tape daily and store the tape off-site.

Daily archives may not be practical for very large databases. To solve this problem, you can activate after-change journals and take a daily archive of the journal itself, which provides archived copies of all changes made since the last full database archive.

Full backup tapes along with the journal backup tapes could be used to restore the entire database.

Effects of Altering Tables on Journals

Use the ALTER TABLE statement to change the columns and the uniqueness of a PI or PPI, and to change, add, remove, or revalidate the partitioning of a PPI. These alterations affect the structure of a table, which causes the table version number to increment.

The ARC utility cannot execute a cluster restore, single AMP restore, or rollforward or rollback of a permanent journal if the current version of a table is different from the archived version number. (For further details, see "Using ARC With Modified PIs or Partitioning" on page 253.)

Journal Subtables

Each journal table consists of three subtables:

- Active subtable
- Saved subtable
- Restored subtable

The active and saved subtables together are referred to as the Current Journal. The restored subtable is called the Restored Journal.

These journal subtables are maintained in an internal Teradata Database format. They are not accessible by SQL statements and cannot be used for audit trail purposes. The following table lists the contents and purpose of each journal.

Subtable	Contents and Purpose
Current Journal	Each time you update a table that has an associated journal table, a change image is appended to the active subtable. You cannot archive journal tables while the change images are in the active subtable. Instead, you must move the images to the saved subtable. To move images from active to saved areas, use the ARC utility and enter the CHECKPOINT WITH SAVE command. A checkpoint places a marker at the chronological end of the active subtable. The database assigns an event number any time a user submits the checkpoint statement. The WITH SAVE option of the CHECKPOINT command inserts a checkpoint in the active subtable and then appends the contents of the active subtable to the end of the saved subtable. After the database appends the contents of the active subtable to the end of the saved subtable, it initiates a new active subtable automatically. You can now submit an ARCHIVE JOURNAL TABLE command.
Restored Journal	To restore the journal, move the journal table contents from the portable storage media back to the restored subtable using the ARC utility. The information stays in the restored subtable until you invoke roll operations.

Permanent Journal Archive or Recovery

To perform archive and recovery functions associated with permanent journals, run the ARC utility. You can execute ARC on a channel-attached host or Windows client. Or you can use an Open Teradata Backup solution. For more information, see Chapter 9: “Archiving, Restoring, and Recovering Data.”

The following table describes archive and recovery commands.

Command	Description
ROLLFORWARD	Replaces a data row by its after-change image, starting from the beginning of the journal and proceeding to either a checkpoint or the end of the journal.
ROLLBACK	Replaces a data row by its before change image from the end of the journal, to a checkpoint or to the beginning of the journal.
DELETE	Deletes the contents of either the saved or restored journal areas.

Backing up tables on Teradata Database involves the following steps:

1. Archive the data tables onto portable storage media.
2. Submit a checkpoint with a SAVE statement to move change images from the active journal to the saved journal.
3. Archive the journal tables onto portable storage media.
4. Submit the DELETE JOURNAL statement to erase the saved journal rows.

Using ARC Statements in a Batch Job

To use ARC statements when a batch program is run:

1. Submit an SQL Checkpoint statement as the first statement of the batch job, with or without a Checkpoint name.
2. If required, ROLLBACK to the Checkpoint using either the checkpoint name or the event number supplied by the DBC when you executed the Checkpoint command. Subsequent changes are also backed out.
3. The data table is now in its *original* condition.

   Note: A permanent journal is time-oriented, not transaction-oriented.

Location of Change Images

Tables that include fallback and journaling options automatically receive dual image journal protection. Tables with no-fallback protection can request either single or dual permanent journals.

The placement of permanent journals depends on the:

- Requested image type (either before or after)
- Protection type (either fallback or no-fallback)

This type of AMP...	Holds these types of images...
Primary AMP	before- and after-image rows for any table with fallback protection. Holds single before images and dual after images for non-fallback protected tables.
Fallback AMP	before- and after-image rows for tables with fallback protection. The system distributes duplicate data rows to fallback processors by assigning the hash code of the row to a different AMP in the cluster.
Backup AMP	single or dual after images and dual before images. Does not use a hashing algorithm for row distribution. All images for one AMP go to a single backup, which is always in the same cluster. For example, if AMPs 1, 2, 3, and 4 are in the same cluster, 1 backs up 2, 2 backs up 3, 3 backs up 4, and 4 backs up 1. There is no way to predict the backup AMP.

If fallback protection is too costly in terms of storage space, after-image journals offer alternative data protection with minimal space usage. After-image journals write changes to the backup AMP. Since the system only duplicates changed rows rather than all of the rows, storage space is minimized.

Since changes are written to the backup AMP, a primary AMP failure does not cause a loss of data. You can recover all table data by restoring the appropriate archive tape and rolling forward the rows stored in the after-image journal.

Creating or Deleting a Permanent Journal

Permanent journals (PJs) are optional. You can specify journal options at the database/user level or at the individual table level.

You create a new permanent journal in the CREATE/MODIFY DATABASE or USER definition. To create permanent journals within an existing user or database, use the MODIFY USER/DATABASE statement.

Unless you specify otherwise at the table level in the CREATE or ALTER TABLE definition, in each table created within that database or user space writes to that journal by default.

You can associate an individual table in that database with a journal in a different database by specifying the fully qualified name of journal (in the form *databasename.journaltablename*) in the CREATE TABLE or ALTER TABLE statement.

Caution: If a database or user that contains a PJ runs out of space, all updates to tables that write to that journal will abort.

The following restrictions apply to the use of PJs:

- If a journal table in another user or database is specified as the default, that other journal table must already exist.
- You can change a DEFAULT JOURNAL for a user or database only if no tables or other databases journal into it.
- PJs are not supported across an AMP configuration change. Rollforward or rollback operations terminate if there is a change in the hash maps for primary, fallback, or backup rows.
- PJs are not supported across certain DDL statements. Statements that may prevent a rollforward or rollback operation from passing that point in the journal include:
  - ALTER TABLE (especially with REVALIDATE PRIMARY INDEX)
  - RENAME TABLE
  - MODIFY USER or MODIFY DATABASE
  - COMMENT

Before you delete a journal, use the ALTER TABLE statement to stop journaling. Use MODIFY USER/DATABASE to remove the journal table.

Assigning a Permanent Journal

Users activate permanent journaling by including the JOURNAL option in the CREATE or MODIFY statements for users or databases. You must allocate sufficient permanent space to a database or user that will contain permanent journals.

Caution: If a database or user that contains a permanent journal runs out of space, all updates to tables that write to that journal abort.

User-Defined Functions (UDFs)

A User-Defined Function (UDF) allows authorized users to write external functions as defined by the ANSI SQL-99 standard. The Teradata Database allows users to create scalar functions to return single value results, aggregate functions to return summary results, and table functions to return tables.

Overview

To develop UDFs, you must:

- Have access rights to create the UDF. Then you can submit the CREATE or REPLACE FUNCTION statement. The system validates and compiles the UDF source as part of the CREATE statement. For more information on privileges, see "Privileges That Must Be Explicitly Granted" on page 165.
- Have a C or C++ compiler on the database server on which the UDFs are installed.
- Write the function source in the C or C++ programming language.
- Debug the function outside of the database.
- Place the source code in a directory that BTEQ can access or in a location that the server can access.

You can also use third-party vendor UDFs as precompiled objects without having to supply the source code.

Note: If you are using ODBC, JDBC, or OLE DB, the source code must be on the server to create the UDF.

To allow functions to access resources, such as local or network files, design the UDF to run in secure mode. In secure mode the UDF is assigned to use a specific OS userid authorization. To accomplish this, first specify the EXTERNAL SECURITY DEFINER/INVOKER clause in the CREATE FUNCTION statement. Then, assign the function to an OS user using the CREATE AUTHORIZATION statement. Any OS user that is given authorization must belong to the "tdatudf" group.

For more information on UDFs, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming* and *SQL Reference: Data Definition Statements*.

For information on how to distribute, backup, and restore packages, see *SQL Reference: Stored Procedures and Embedded SQL*.

Compiling UDFs

The system reports compilation errors and links the function object into a dynamically linked library if there are no errors. The dynamically linked library is then distributed to all the nodes in the system and the UDF is usable as soon as the CREATE statement completes.

If the C or C++ compiler on the client system is compatible with the C or C++ compiler on the server, and the generated objects are also compatible, Teradata strongly recommends that you develop, compile, and debug UDFs on the client system. This is because there are no debugging facilities for a UDF installed in the database other than a callable trace facility.

If the compiler is not compatible or there is no compiler on the client system, then the UDF developer must obtain access from the DBA to place the UDF files, both source and object files, on the process distribution node (node 0) of the server. The UDF files should reside in the source and object file directory location (see "Directory Paths for UDFs" on page 85) on this node so that the system can copy and use them upon execution of the CREATE/REPLACE FUNCTION statement. The DBA may allow UDF developers to access the server however they choose, for example, by using FTP to copy the files up to the server.

Regardless of the method, a copy of the header files used by the system to compile the function needs to be made available. To develop and test the UDF outside the database, a copy of the header file, sqltypes_td.h, must be specified with a C include directive in the source when UDFs are created.

If you are using ...	The header file sqltypes_td.h should be located at...
Windows	C:\Program Files\NCR\TDAT\LTDBMS\etc
UNIX MP-RAS	/tpasw/etc
Linux	/usr/tdbms/etc

Note: The database server on which UDFs are developed must have a C compiler. All systems require a C compiler on at least one PE node even if the function is distributed as an object or a linked library. The compilers used for UDFs are the same as for Stored Procedures.

Testing UDFs

To protect the system from untested functions, the system defaults to the "protected" execution mode. This means that the system isolates all of the data the UDF might access as a separate process in its own local workspace. If any memory violation or other system error occurs, the error is localized to the function and the transaction executing the function. Running under protected execution mode makes the function run slower. However, once the function has been tested and validated, the execution mode can be changed to non-protected mode using the ALTER FUNCTION statement. The function does not have to be recompiled or re-linked. For more information, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.

Creating a Database for General Global Functions

The system will use a database called "SYSLIB" to place general global UDFs in case a function is called without a qualifying database name. When the calling function does not qualify a database name, the system first searches for the UDF from the current default database. If it does not find the UDF in the default database, the system looks in SYSLIB.

You can create the SYSLIB database yourself or use the DIP script.

Note: The DIPPATCH option in the DIP script creates SYSLIB as part of dipview.bteq and with zero permanent space. Use MODIFY DATABASE SYSLIB AS PERM = <*desired_number*> to increase the permanent space before creating any UDFs in this database. Also, you must grant the appropriate access privileges on SYSLIB before creating any UDFs in it.

Placing UDFs in SYSLIB is also one way of making them accessible to users who have been granted the EXECUTE FUNCTION privilege on the UDFs or the SYSLIB database.

Controlling Access of UDFs

Previous to V2R6.1, you could only run protected mode UDFs under the "tdatuser." Now, using the CREATE AUTHORIZATION…AS DEFINER/INVOKER statement, you can run UDFs under any OS user on the system so that the UDF can access whatever that OS user has privileges to access.

Warning About Altering System Objects

As part of the installation process, the tdatuser user and tdatudf group are created on each node the database runs on. In order for UDFs to function correctly and for you to be able to properly create UDFs, do not alter any of the objects listed in the following table.

Object Name	Object Type	Description
tdatuser	user	This system user helps run UDFs in protected mode. **Do not** attempt to: • Delete tdatuser • Change the password on W2K or • Add a password on MP-RAS (this prevents logon to the user on UNIX systems). tdatuser has no special privileges on the system. Do not try to log in as this user. No one should be able to log on as tdatuser. On MP-RAS, tdatuser has a */home/tdatuser* directory which is used for "core" dumps of protected mode UDFs that crash.
tdatudf	group	A group called 'tdatudf' is created on each node. Do not make any changes to this group.

For more information on tdatuser and tdatudf, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*. If you feel you need to make changes to tdatuser or tdatudf, please consult Teradata Support Center first.

Directory Paths for UDFs

The following table lists the files and default path locations used by UDFs.

UDF File/ Directories	Windows	UNIX	Linux	Description
Header file	C:\Program Files\NCR\Tdat\LTDBMS\etc	/tpasw/etc	/usr/tdbms/etc	The header file, sqltypes_td.h, must be specified with a C include directive in the source when UDFs are created. This file can be copied if the UDF developer compiles or develops the UDF outside of the database.
Source and object directory path	C:\Program Files\NCR\Tdat\tdconfig\Teradata\tdbs_udf\usr	/Teradata/tdbs_udf/usr	/etc/opt/teradata/tdconfig/Teradata/tdbs_udf/usr/	This directory is the default location the system searches for source and object files. **Note:** The DBA, or someone with sufficient privileges, is responsible for manually creating this directory. When you create a UDF and do not specify the location of the source or object file, or you specify a relative path for the source or object file, the full path begins in the default directory.
Compiler temporary path	C:\Program Files\NCR\Tdat\TdTemp\UDFTemp\	/tmp/UDFTemp/	/var/opt/teradata/tdtemp/UDFTemp/	The temporary directory where UDFs are compiled. (Any files needed for the compilation process are moved here. This includes source files from the server or client as well as object and header files, if needed.) The temporary compile directories only exist for the duration of the compile.
UDF library path	C:\Program Files\NCR\Tdat\tdconfig\udflib\	/ntos/udflib/	/etc/opt/teradata/tdconfig/udflib/	The directory where the dynamically linked libraries are saved.
UDF server memory path	C:\Program Files\NCR\Tdat\TdTemp\udfsrv\	/tmp/udfsrv/	/var/opt/teradata/tdtemp/udfsrv/	The directory where the shared memory files used for the execution of protected mode UDFs reside.

The paths in the preceding table are default locations. If you installed Teradata Database on a Windows or Linux system, and did not use the default paths for installation, configuration, temporary and dump files, substitute paths in the preceding table accordingly.

For example, on Windows, the temporary folders path for Teradata Database (<TdatTemp>) is usually located at C:\Program Files\NCR\TDAT\tdconfig. Although this is the usual Windows path, be sure to check if the path was set differently during system installation.

To get the configuration path, enter "pdepath -c" on the command line.
To get the path where etc files are located, enter "pdepath -e" on the command line.

Additional paths for Linux include:

- CompilerPath: /usr/bin/gcc
- Linker Path: /usr/bin/ld
- Teradata Stored Procedure Library Base: /etc/opt/teradata/tdconfig/tdsplib/

For more information on how to create, compile, and use UDFs, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming.*

User-Defined Types (UDTs)

User-Defined Types (UDTs) allow you to create custom data types that model the structure and behavior of data in your applications.

Overview

There are two kinds of UDTs:

- A distinct UDT takes on all of the characteristics of an already existing data type such as BYTE, DECIMAL, VARCHAR, etc. It is basically a redefinition of an already pre-defined SQL data type.
- A structured UDT is a collection of fields called attributes that define either a pre-existing data type or another UDT (which allows nesting).

UDTs reside in the SYSUDTLIB database. Initially, only user DBC has implicit privileges on the SYSUDTLIB database. For any other user or role to interact with SYSUDTLIB, they must first be explicitly granted UDT privileges (UDTUSAGE, UDTTYPE, or UDTMETHOD) on SYSUDTLIB to execute SQL statements that involve UDTs. For more information on UDT privileges, see user-defined types in the table under "Description of Privileges" on page 167.

For using UDTs with client products such as MultiLoad or TPump, you must specify the underlying pre-defined data for distinct UDTs and specify the client/host representation or UDT attributes for structured UDTs. For more information on loading tables using UDTs with client utilities, see the respective load utility manual.

For more information and a full description of the restrictions, see "CREATE TYPE" in *SQL Reference: Data Definition Statements.*

Restrictions for UDTs

Some restrictions for using UDTs include:

- No UDT can be part of an index definition.
- The first column on a table cannot be a UDT.
- A table containing UDT columns must have at least one non-UDT column.
- A UDT must have both ordering and transform defined prior to its usage as a column definition.
- It is highly recommended that if the type developer intends to support null attributes or authors a map ordering routine that can return null, then all of the CREATE/ALTER TABLE statements that utilize that type should have the NOT NULL phrase present on that UDT column.
- You can use UDTs as parameters or return values for stored procedures and external stored procedures in C/C++ but not for Java external stored procedures.

In addition, the total number of UDT columns you can define depends on the available table header space. For a full discussion on table header space restrictions, see "Table Header Format" in *Database Design*.

Creating a Database for UDTs

The system uses database "SYSUDTLIB", which is automatically created by the DIP script, to store all routines and methods required for creating and using UDTs as well as the UDTs themselves. SYSUDTLIB initially has no space allocated to it so you must add space to accommodate new UDTs and associated UDFs and UDMs.

Note: Do not store non-UDT related data in SYSUDTLIB because it needs all its space for UDTs and associated methods and routines.

Only user DBC has implicit privileges on the SYSUDTLIB database. For any other user or role to interact with SYSUDTLIB, they must first be explicitly granted UDT privileges (UDTUSAGE, UDTTYPE, or UDTMETHOD) on SYSUDTLIB to execute SQL statements that involve UDTs.

Required Space for UDTs

Teradata Database requires space for every UDT defined and for every method or UDT-related UDF created with it. The initial space allocation for the SYSUDTLIB database is zero. You must alter this table and assign space in order for UDTs, UDMs, and UDT-related UDFs to be created.

Use the MODIFY DATABASE SYSUDTLIB AS PERM = <*desired_number*> statement to increase the permanent space for SYSUDTLIB.

In addition, from time to time, you will need to analyze your system and increase space for SYSUDTLIB as the number of UDTs, UDMs, and UDT-related UDFs grows.

Controlling Casting Behavior

The system implicitly converts a UDT expression to a compatible predefined type when there is a CAST…AS ASSIGNMENT definition whose target is a compatible data type.

You can turn off this implicit casting through the DBSControl flag "DisableUDTImplCastForSysFuncOp." This field defaults to FALSE and the system automatically casts UDTs. If the flag is set to TRUE, however, and an attempt is made to pass a UDT to a built-in system function or operator, the system returns an SQL error.

Note: The DisableUDTImplCastForSysFuncOp flag only affects built-in system operators and functions. The system always invokes implicit casts for INSERT, UPDATE, and parameter passing (UDF, UDM, Stored Procedure) operations.

User-Defined Methods (UDMs)

User-Defined Methods (UDMs) are special kinds of UDFs that are associated with a particular UDT. There are two types of methods:

- An instance method operates on a specific instance of a distinct or structured UDT.
- A constructor method initializes an instance of a structured UDT.

UDMs, like UDFs, support protected or non-protected modes of execution and are written in C or C++. UDMs, like UDTs, are always created in the SYSUDTLIB database.

The same privileges required for using UDTs are necessary for creating or using UDMs.

For more information on UDMs, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.

Required Space for UDMs

Teradata Database requires space for every UDT defined and for every method or UDT related UDF created with it. The initial space allocation for the SYSUDTLIB database is zero. You must alter this table and assign space in order for UDTs, UDMs, and UDT-related UDFs to be created. From time to time, you will need to analyze your system and increase space for SYSUDTLIB as the number of UDTs, UDMs, and UDFs grows.

CHAPTER 3 Space Considerations

This chapter describes things to consider for setting up disk space limits and how best to manage space. Space (whether permanent, spool, or temporary) is measure in BYTES.

Overview of Space

The following table lists the three different kinds of space.

Type of Space	Description
Permanent (Perm)	Perm space allocated to a user or database is a uniquely defined, logical repository for: • Data tables • Table headers (two header rows per table) • Secondary indexes (SIs) • Join indexes (JIs) • Hash indexes (HIs) • Permanent journals • Stored procedures • Triggers • User-defined functions (UDFs) • User-defined methods (UDMs) When an object is created or data rows are inserted, the space is allocated as needed from the perm space of the immediate owner. The space is returned automatically when no longer needed. **Note:** Perm space allocated to a user or database that is unassigned to an object is available for spool or temp space. 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.
Spool	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. 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 same amount of spool as its parent. 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.
Temporary (Temp)	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. 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.

Note: You can manage space using Teradata SQL. However, you can also administer space using Teradata Administrator. For more information, see *Teradata Administrator User Guide*.

Defining Permanent Space Limits

The system uses permanent space to store objects like tables, indexes, stored procedures, functions, and permanent journals[1]. You set permanent space limits at the database or user (not table) level and define the maximum limit with the PERM parameter of a CREATE/ MODIFY USER/DATABASE statement. This allocation is deducted from the limit of the immediate parent of the object being created.

The amount of permanent space specified for each user or database is divided by the number of AMPs in the configuration. The result is recorded on each AMP and may not be exceeded on that AMP. In other words, you may create several objects in your own space, as long as the combined storage requirements do not exceed the maximum limit (See MAXPERM) set on each AMP.

Permanent Space Availability

Initially, all available permanent space (or PERM space) in the system is allocated to user DBC (see "The Initial Teradata Database" on page 36). From user DBC, you create other users who use up some of the space owned by DBC. As you create additional new databases or users under those users, permanent space limits are deducted from the available (unused) space of the immediate owner of that database or user. If those users use up all the space allocated to them by their owner, those users cannot create new objects until they obtain more space.

Note: Unused space is allocated dynamically for temporary or spool space, which can reduce the actual amount of perm space available at a given point in time. Thus, specifying a PERM limit does not guarantee that a session can get all of that space upon demand.

To determine the amount of perm space available by a per-AMP basis, query the DBC.DiskSpace view. To obtain information for all AMPs on your system, use the SUM aggregate. For more information, see "DisckSpace[V][X]" in *Data Dictionary*.

Permanent Space Allocation

Permanent space is dynamically acquired by data blocks and cylinders when the system inserts rows. The data block is a disk-resident structure that contains one or more rows from the same table and is the physical I/O unit for the file system. Data blocks are stored in physical disk sectors or segments, which are grouped in cylinders.

The total space for your entire Teradata Database configuration is derived by summing all database and user PERM limits. Total unallocated space is equal to the sum of all permanent limits minus the sum of all permanent space currently in use. (For information on how to track disk space usage, see "Obtaining Disk Space Information" on page 100.)

You can use the DBC.DiskSpace view to determine how much permanent space is being allocated to which database or user. The following table lists the fields that track perm space values.

[1] Triggers, macros, and views do not take up any space in their databases.

Type	Description
CURRENTPERM	The total number of bytes (including table headers) currently allocated to existing data tables, index tables and subtables, stored procedures, triggers, and permanent journals residing in a particular database/user. This value is maintained on each AMP.
MAXPERM	The maximum number of bytes available for storage of all (current and future) data tables, index tables and subtables, stored procedures, triggers, and permanent journals owned by a particular database/user. **Note:** For each database or user reported, the number of permanent bytes is divided by the number of AMPs in the configuration. The result is recorded on each AMP and may not be exceeded on that AMP. That is, a user may own several objects, as long as the combined storage requirements are within the MaxPerm limit set for that user on each AMP.
PEAKPERM	The largest number of bytes ever used to store data in a user or database since the last reset of this value to zero. This value is maintained on each AMP. To reset the PeakPerm value to zero, use the DBC.ClearPeakDisk macro. For more information on this macro and how to use it, see "Resetting Peak Values" on page 105 and *Data Dictionary*.

For more information about perm space allocation (including the ramifications of space allocation for users DBC, SystemFE, and SysAdmin), see "Permanent Space Allocations" in *Database Design*.

Reserving Cylinders for Permanent Space Allocation

A transaction is aborted when it requires more space than is currently available for the requesting user. To protect transactions, you can specify that a number of cylinders be reserved for transactions needing permanent space.

When you create a user or database, the system requests space on cylinders from the pool of free cylinders. A cylinder logically holds either perm or spool but not a combination. (Because a cylinder is logical, Teradata Database does not require contiguous disk space and the physical disk space is allocated only as needed when rows are actually inserted into tables.)

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 the DBS Control Record (see DBSControl Utility in *Utilities*). 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.

Setting Data Block Size Limits

In the Teradata Database file system, a data block is a segment of one or more data rows preserved on disk. This segment contains the actual data rows, all of which are members of the same subtable. Any single data row is fully contained within a single data block. A row that exceeds the size of a multi-row data block is put in a block of its own.

Every data block must be fully contained within a cylinder. The cylinder is a group of consecutive disk sectors (also called segments) that is not necessarily identical to a physical disk cylinder. The file system normally uses an averaged cylinder size, and each cylinder is logically independent from other cylinders.

Set the maximum size of data blocks using:

- The DBS Control utility. The PermDBAllocUnit, PermDBSize, and JournalDBSize fields of the DBS Control record are global parameters that determine the maximum size of permanent data blocks that hold multiple rows. (A row that exceeds the size of a multi-row data block is put into a block of its own.) You can override this value at the table level.

  Note: The default and minimum data block size depends on the cylinder size your site uses. Cylinder size is a global value, shown as:

  - SectsPerCyl (with the Filer Utility)
  - Number of Disk Blocks Per Cylinder (with the PUT utility)
- 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 to be used when storing rows of this table. To improve performance, the row length should be short enough to write two or more rows to one data block. The maximum value possible for a datablock is 127.5 Kbytes.

Setting Free Space Percent Limits

Free Space Percent is the percentage of cylinder space to leave free during load operations. You can control this space at the global and table level as described in the following table.

Parameter	Description
FreeSpacePercent of the DBS Control record	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. Choose a percentage that reflects the net growth rate of your data tables (INSERTs minus DELETEs). Most sites choose 5 to 15 percent. Run the PACKDISK command of the Ferret utility after setting this value to get data packed to the specified percent.
"Cylinders Saved for PERM" field of the DBS Control record	Used to save some number of cylinders for permanent data only. **Note:** If the number of free cylinders falls below this value, any allocation of cylinders for spool data results in an abort of the requesting transaction.
FREESPACE = *n* of an CREATE/ALTER TABLE statement	*n* percent of cylinder space to remain free on each cylinder when bulk loading this table (where *n* is an integer constant).
DEFAULT FREESPACE of an ALTER TABLE statement	Reset the current freespace percent for a table at the global or table level. DEFAULT FREESPACE resets Free Space to the value defined in the FreeSpacePercent field of the DBS Control record.

For more information on the DBS Control record, see "DBSControl (dbscontrol)" in *Utilities*. For information on the CREATE TABLE or ALTER TABLE statements, see *SQL Reference: Data Definition Statements*.

Specifying Spool Space Limits

Spool space is permanent space not assigned to any database or user and remains spool until permanent space is required.

The system needs spool space for TJs or to create spool files for processing queries when returning an answer. Spool space is especially important for queries that involve full table scans or use NUSIs. Because spool space is critical to the operation of Teradata Database, be sure not to overlook capacity planning for it.

Spool Space and Capacity Planning

The system uses spool space for the following:

- The response rows of every query run by that user during a session. Thus, each user needs a high enough spool allocation to contain the biggest anticipated answer set.

- Very large tables. Large tables usually require more available spool space than smaller tables, because intermediate rows are held in spool space during query execution.
- Large volatile tables. These tables require more available spool space.

A rough estimate for required spool for your entire system is 30 percent of your MAXPERM; this estimate is for a system with no fallback, no compression, and a medium workload. Or stated another way, spool for a no fallback, no compression, medium workload system should be about 40 percent the size of CURRENTPERM. Some workloads will need more or less spool. When deciding on how much spool to allocate for your system, consider the following:

Requires More Spool Space	Requires Less Spool Space
Highly compressed data requires more spool. Compression and multi-value compression decreases the current perm data storage but does not always decrease the spool space because spool tables are not always also compressed. **Note:** There are some instances when values in spool are compressed but you should still be generous with required spool space estimates and assume compression on permanent data will not mean compression on spool data.	Fallback doubles the currentperm data storage but does not double the spool space requirement because only the primary copy of the data will be in spool.
Batch loads can load millions of rows or more at a time and require spool space to accommodate temporary disk storage required for intermediate load processing steps.	Real-time loads, such as with TPump, load only a few rows at a time, and do not require significant spool space.
High concurrency (aggravated when large spool queries running at a low Priority Scheduler priority) and large intermediate spools (which use a lot of temporary tables).	Well-designed and properly tuned queries use less CPU as well as less spool.

Size requirements vary from user to user, table to table, and application to application. Some sites only need 20 percent of system space set aside for spool while others may require 50 percent: your Teradata support personnel can assist you in determining a good value with which to start.

Because every site is different and requirements can change from time to time, it is a good idea to regularly re-assess and re-allocate spool when the configurations or workloads on the system change.

Note: The system divides space by the number of AMPs. For example, if you have 4 AMPs, you would divide the total space given to DBC by 4. Each AMP gets 25 percent of the space available. This includes spool space.

Types of Spool Space

Spool falls into three categories of space: volatile, intermediate, and output.

Type	Description
Volatile Spool	Volatile spool is retained until the: • Transaction completes (unless the table was created with ON COMMIT PRESERVE ROW) • Table is dropped manually during the session • Session ends • Teradata Database resets
Intermediate Spool	Intermediate spool results are retained until no longer needed. You can determine when intermediate spool is flushed by examining the output of an EXPLAIN. **Note:** The first step performed after intermediate spool has been flushed is designated "Last Use."
Output Spool	Output results are either: • Final rows returned in the answer set for a query • Rows updated within, inserted into, or deleted from a base table The length of time spool space is retained for output results depends on a variety of conditions.

Spool Space Allocation

Teradata Database allocates spool space dynamically only from disk cylinders that are not being used for permanent or temporary data.

Note: Permanent, temporary, and spool data blocks cannot coexist on the same cylinder.

Spool space is not reserved. All unused perm space in the Teradata Database is considered available spool space. When spool is released, the file system returns the cylinders it was using to the free cylinder list. To reserve spool, create a spool reserve database. For more information, see "Creating a Spool Reserve Database" on page 97.

Limiting spool space for a user helps reduce the impact of possibly bad queries. You set spool space limits for a database, a user, or use a profile to limit spool space for a user, but you cannot limit spool space at the table level. (For more information about how to set up and assign profiles to users, see CREATE USER and CREATE PROFILE in *SQL Reference: Data Definition Statements*.)

The maximum and default limits for database, user, and profile spool space are determined as described in the following table.

IF you ...	AND a profile...	THEN the limit...
specify SPOOL in a CREATE/MODIFY USER/DATABASE statement	does not apply	may not exceed the limit of the immediate owner of the user or database.
	applies	may not exceed the limit of the user who submitted the CREATE/MODIFY PROFILE statement, determined as follows: • If that user has a profile, the limit in the profile • If a profile does not apply to that user, the limit in the CREATE/MODIFY USER statement for that user. • If no SPOOL is defined for that user, the limit is the same as the immediate owner.
do not specify a SPOOL limit in a CREATE/MODIFY USER/DATABASE statement	does not apply	is inherited from the specification for the immediate owner of the user or database.
	applies	is inherited from the profile specification.
	applies but the SPOOL parameter is NULL or NONE	for the user who submitted the CREATE/MODIFY PROFILE statement, determined as follows: • 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 SPOOL is defined for that user, the specification for the immediate owning database or user.

Empirically Determining Spool Requirements

To determine your spool size requirements based on empirical trends, use the Data Collection facility of Teradata Manager to collect spool use history on a per system basis. (You can also do it on a per-user basis for added detail.) The data collection feature allows you to specify usage collected by user, amount of time the summary data is kept, the amount of spool usage allowable before an alarm action is triggered, and more.

After collecting spool usage data, you can use Teradata Manager's trend reporting tool to view the results graphically or in table form. Get historical use of spool usage from these reports and adjust your spool size needs according to the empirical values shown in the trends. For more information, see *Teradata Manager User Guide* or the online help of Teradata Manager.

Reserving Minimum Spool Space

Teradata 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 percent of space relative to CURRENTPERM. (For example, if CURRENTPERM is 100GB, spool space will be 40GB for a total MAXPERM of 140GB.)

Although, 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.

IF your system has the following fallback rate...	AND the following compression rate...	THEN consider reserving the following spool space percentage[a]...
100%	0%	20%
100%	40%	30%
50%	20%	32%
0%	0%	40%
0%	40%	60%

a. This percentage is relative to CURRENTPERM.

Note: Because every site is different and can greatly vary, use the numbers above as guidelines.

Creating a Spool Reserve Database

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 Teradata Database as user DBC.
2. Submit the following statement:

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

   where *xxxx* is the number of bytes that is a specific percentage of your total available space as outlined in "Reserving Minimum Spool Space" on page 97.
3. Quit the session and log off.

For more information on determining the allocation of spool space, see "Sizing Spool Space" in *Database Design*.

Limiting the Spool Space for a User

Limiting the amount of spool space you allocate to a user helps reduce the impact of "runaway" transactions such as accidental product joins. A bad query with a highly nonunique index or a missing close parenthesis in its code would run out of spool space before tying up system resources.

When using the CREATE USER, CREATE DATABASE, or CREATE PROFILE statements to assign spool space limits, keep the following space levels in mind.

Spool Space Level	Description
MAXSPOOL	MaxSpool is a value used to limit the number of bytes the system allocates to create spool files for a user. The value you specify may not exceed the value of the immediate parent (database or user) at the time you create the user. If you do not specify a value, MaxSpool defaults to the MaxSpool value of the parent. Specify the spool limit of each user, taking into consideration the tables they access, to reduce the impact of "runaway" transactions, such as accidental Cartesian product joins.
CURRENTSPOOL	CurrentSpool is the number of bytes in use for resolving queries. This value is maintained on each AMP for each user.
PEAKSPOOL	PeakSpool is the maximum number of bytes used by a transaction for a user since the value was last reset by the ClearPeakDisk Macro (supplied in user DBC). See “Resetting Peak Values” on page 105 and *Data Dictionary* for more information on this macro.

You can query the DBC.DiskSpace view to find the system levels for spool space.

Defining Temporary Space Limits

Temporary space is used to hold rows of materialized global temporary tables. It is allocated at the database or user level, but not the table level.

You 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 temp 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: • 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: • 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.

Query the DBC.DiskSpaceX view to find the system levels for temporary space. 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.DiskSpaceX 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.
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 tables. The value may not exceed the limit of: • The creator or modifier of the profile, when setting TEMPORARY in a profile • The immediate owner of the user being created or modified, if a profile does not apply 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.

Obtaining Disk Space Information

The following sections describe some system views you can use to determine your current space usage and allocation. For information on all the system views, see *Data Dictionary*.

DiskSpace View

The DBC.DiskSpace view returns AMP information about disk space usage at the database or user level. It also can report spool space usage. DiskSpace 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.DiskSpaceX. By querying MaxSpool and MaxTemp, you get the space limits for your individual user space. See the *Data Dictionary* for a list of all the possible fields in DBC.DiskSpace.

The following queries show the kinds of information you can obtain from DBC.DiskSpace.

TO find the...	SUBMIT the following query...
total disk space in the entire system	`SELECT SUM(MaxPerm) FROM DBC.DiskSpace;`

disk space currently in use	`SELECT SUM(CurrentPerm),`
	`       SUM(MaxPerm),`
	`       ((SUM(currentperm) / NULLIFZERO(SUM(maxperm)) * 100))`
	`       (TITLE '%MaxPerm', FORMAT 'zz9.99')`
	`FROM DBC.DiskSpace;`

disk space for a given database	`SELECT sum(maxperm`
	`FROM DBC.DiskSpace WHERE databasename='xxxx';`

percent of disk space that is available for spool	`SELECT (((SUM(MaxPerm) - SUM(CurrentPerm)) /`
	`NULLIFZERO(SUM(MaxPerm))) * 100)`
	`(TITLE'% Avail for Spool', format'zz9.99')`
	`FROM DBC.DiskSpace;`

percent of space used by each database in the system	`SELECT Databasename (format 'X(12)')`
	`,SUM(maxperm)`
	`,SUM(currentperm)`
	`,((SUM(currentperm))/`
	`NULLIFZERO (SUM(maxperm)) * 100)`
	`(FORMAT 'zz9.99%', TITLE 'Percent // Used')`
	`FROM DBC.DiskSpace`
	`GROUP BY 1`
	`ORDER BY 4 DESC`
	`WITH SUM (currentperm), SUM(maxperm);`

users who are running out of perm space	`SELECT Databasename (format 'X(12)')`
	`,SUM(maxperm)`
	`,SUM(currentperm)`
	`,((SUM(currentperm))/`
	`NULLIFZERO (SUM(maxperm)) * 100)`
	`(format 'zz9.99%', TITLE 'Percent // Used')`
	`FROM DBC.DiskSpace`
	`GROUP BY 1 HAVING (SUM(currentPerm) / NULLIFZERO(SUM(maxperm))) > 0.9`
	`ORDER BY 4 DESC;`

users using a lot of spool	`SELECT databasename`
	`,SUM(peakspool)`
	`FROM DBC.DiskSpace`
	`GROUP BY 1 HAVING SUM(peakspool) > 5000000000`
	`ORDER BY 2 DESC;`
	Note: You can change the value 500000000 to whatever value is appropriate for your site. Some sites with more space may have higher tolerance for higher spool usage and spool limits.

For the latest example queries for obtaining space usage information, see Knowledge Article CA9001181E6 located in the Teradata Database Knowledge Repositories. This article is available through Teradata @ Your Service. Sign up for an account at www.teradataatyourservice.com.

TableSize View

The DBC.TableSize view provides AMP information about disk space usage at the table level. Optionally use the TableSizeX 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	`SELECT tablename (TITLE 'Table')`
	`,currentperm (TITLE 'CurPerm')`
	`,vproc (TITLE 'Amp')`
	`FROM DBC.tablesize`
	`WHERE databasename='xxx'`
	`AND tablename = 'xxxx'`
	`ORDER BY 2 DESC;`

disk space for a given table	`SELECT SUM(currentperm)`
	`FROM dbc.tablesize`
	`WHERE databasename='xxx'`
	`AND tablename = 'xxxx';`

If you submit a SELECT statement against DBC.TableSize and notice it takes a long time to process, replace the view definition by submitting the following query:

```
REPLACE VIEW DBC.TableSize
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.

Example of Finding Skewed Tables

In this example, the SELECT statement looks for poorly distributed tables by displaying the CurrentPerm figures allocated on each AMP to every table in the User database. Each table is reported separately, ordered by name and by AMP.

```
SELECT vproc AS AMPno, TableName (FORMAT ’X(20)’), CurrentPerm
FROM DBC.TableSize
WHERE DatabaseName = ’USER’
ORDER BY TableName, AMP ;
```

The result displays two tables. Table Employee is evenly distributed across all AMPs in the system. The CurrentPerm figures range from 4,096 bytes to 30,208 bytes on different AMPs.

Notice the results show that:

- CurrentPerm is similar across all vprocs for Employee_upi_onempid
- The table Employee_nupi_ondept is poorly distributed

```
AMPno    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_ondept         4,096
1        employee_nupi_ondept         30,208
2        employee_nupi_ondept         15,360
3        employee_nupi_ondept         12,288
```

Warning About AllSpace View

The DBC.AllSpace 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.DiskSpace and DBC.TableSize views instead since DBC.AllSpace can return misleading results.

For example, the following query returns the current perm from the detail rows *and* summary rows for each user or database. Note that the view sums data from both types of rows:

```
sel databasename
        ,sum(currentperm)
     from DBC.AllSpace
     group by 1 having sum(currentperm) > 0
     order by 2 desc;
```

```
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.DiskSpace will return the desired result and, on most systems, will run faster:

```
     sel databasename
        ,sum(currentperm)
     from DBC.DiskSpace
     group by 1 having sum(currentperm) > 0
     order by 2 desc;
```

```
DatabaseName                       Sum(CurrentPerm)
------------------------------  -------------------
DBC                                      23,591,936
Sys_Calendar                              1,324,032
SysAdmin                                    690,176
SYSLIB                                      159,744
SystemFe                                     74,240
```

The view DBC.DiskSpace includes only summary rows and the view DBC.TableSize includes only detail rows. The view DBC.AllSpace, however, includes both summary and detail rows and when you sum currentperm fro AllSpace, 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.DiskSpace view which selects only summary (user/database) rows. If you want space usage information at the object level, use the DBC.TableSize view.

Comparing DBC.DiskSpace and DBC.TableSize Results

The following space accounting views report different values.

TO find the available space at ...	THEN use this view ...
the database level.	DBC.DiskSpace
the table level.	DBC.TableSize

DiskSpace reports on a database while TableSize reports on the tables within a database. Thus:

- TableSize reports less MAX(CurrentPerm) than DiskSpace.
- Both views report the same SUM(CurrentPerm).

This difference is explained in the examples and results comparison below.

Example

Use TableSize to report the perm of tables in a particular database and DiskSpace to report the perm of the database.

- TableSize reports MAX(CurrentPerm) and SUM(CurrentPerm) on all tables in the specified database. For example, assume you enter the following:

  ```
  SELECT MAX(CurrentPerm),SUM(CurrentPerm)
  FROM DBC.TableSize
  WHERE DatabaseName = MARKETING ;
  ```

  For the purposes of our example, the space is reported as follows:

  ```
  MAX(CurrentPerm)  SUM(CurrentPerm)
  ----------------  ----------------
  4,096             17,408
  ```

- DiskSpace reports MAX(CurrentPerm) and SUM(CurrentPerm) at the database level:

  ```
  SELECT MAX(CurrentPerm),SUM(CurrentPerm)
  FROM DBC.DiskSpace
  WHERE DatabaseName = MARKETING ;
  ```

  For the purposes of our example, the response is as follows:

  ```
  MAX(CurrentPerm)     SUM(CurrentPerm)
  -----------------    -----------------
  6,144             17,408
  ```

The following table compares the amounts returned by the CurrentPerm column of the DiskSpace and TableSize views.

Space	DBC.DiskSpace View	DBC.TableSize View
SUM(CurrentPerm)	Reports the space consumed by all the tables in the specified database or all databases. This amount should agree with the SUM(CurrentPerm) reported by TableSize.	Reports the total for all user tables in the specified database or all databases. The total is found by adding together the bytes consumed by each table. This amount should agree with the SUM(CurrentPerm) reported by DiskSpace.
MAX(CurrentPerm)	Returns results at the database level. MAX is the PERM defined for the database/user in the CREATE or MODIFY statement.	Returns the remainder of defined PERM space minus SUM space. This may or may not agree with MAX(CurrentPerm) returned by DiskSpace.

Resetting Peak Values

Each of the system views listed previously are views of the non-hashed DBC.DatabaseSpace table. From time to time, you need to clear out the peak values accumulated in the DBC.DataBaseSpace table to restart the data collection process.

Teradata provides the DBC.ClearPeakDisk macro to reset the PeakPerm and PeakSpool values in the DBC.DataBaseSpace table. To review the definition of ClearPeakDisk, enter:

```
SHOW MACRO DBC.ClearPeakDisk ;
```

Teradata Database returns the contents of the CREATE MACRO DDL:

```
REPLACE MACRO ClearPeakDisk AS (
UPDATE DatabaseSpace
SET PeakPermSpace = 0,
PeakSpoolSpace = 0 ALL ;) ;
```

Before invoking the macro, you may want to save the peak values, either in a separate collection-periods table or offline.

To zero out the PeakPerm and PeakSpool values in the DatabaseSpace table for the next data collection period, enter:

```
EXEC DBC.ClearPeakDisk;
```

Teradata Database returns the number of rows changed:

```
*** Update completed. 3911 rows changed.
*** Time was 4 seconds.
```

Gaining Space with PACKDISK

The PACKDISK command of the Ferret Utility reconfigures the contents of a disk, leaving a percentage of free space for cylinders within a scope defined by the SCOPE command. Ferret is meant to be used only by Teradata personnel. However, if you have contacted Teradata Support Personnel and have received instructions for using Ferret, see *Utilities* for more information on associated commands.

Packing and Free Space Percent

PACKDISK packs either the entire disk or a single table, leaving a specified percentage of the object empty to account for subsequent insert and update operations. This is the free space percentage (FSP) you specify using the tools described in "Setting Free Space Percent Limits" on page 93.

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.

Using SHOWFSP

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 do a defrag.

```
===========================================================
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         Table                           fsp  Recoverable Current
  Name             Name                            %    Cylinders   Cylinders
----------------- ---------------------------  -----   ---------  ---------
CUSTOMER           ppitable3                       25    1           5
vproc  1 (0001)  response

There are  6 tables larger than 0 cylinders on amp 1

 Database            Table                       fsp   Recoverable Current
 Name                Name                        %     Cylinders   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" in *Utilities*.

Increasing Space by Giving Ownership

This section describes how to increase the permanent space limit of a database or user by transferring a database or user to another database or user.

Transferring Ownership

You can transfer both databases or users, but to simplify the discussion, this section describes databases. Note that you can easily replace any instance of the word “database” in the following section with the word “user.” Also note that databases can own users and users can own databases.

The GIVE statement transfers ownership of a database, including all the databases owned by that database. In addition, you transfer the permanent space limits defined for that database.

When you give a database to another database in the hierarchy, only the permanent space limit for that database is transferred. The spool and temporary space limits remain the same (even if they are higher than the new immediate owner). All descendents of the given database remain descendents of that given database.

When you drop a database, its permanent space limit is credited to its immediate owner; however, its spool space and temporary space is not credited to its immediate owner (that is, the immediate spool and temporary space allocation of the owner remains unchanged).

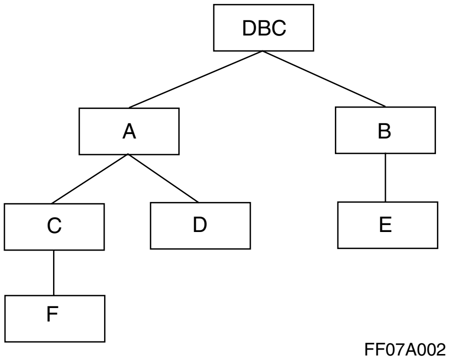
GIVE does not transfer and change explicit privileges. For performance implications, see “Effect of the GIVE Statement on Privileges” on page 182 and the GIVE Statement under “Data Control Language Syntax” in *SQL Reference: Data Definition Statements*.

For information on using ownership to change the current hierarchy, see “Removing a Hierarchy Level” on page 34.

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 wish to transfer permanent space from a child of a child back to User DBC when User DBC is not the immediate owner.

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 temp 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.

For more information on transferring ownership, see "Changing the Hierarchy with GIVE" on page 33.

SECTION 2 Security and Authorization

CHAPTER 4 Using Data Dictionary Tables and Views

The Data Dictionary (the Teradata Database system catalog), is a complete database composed of tables, views, and macros that reside in system user DBC. The Data Dictionary contains information about the entire database.

This chapter introduces you to the concept of the Data Dictionary and how to use and maintain it, including how to use the system provided views to access the Data Dictionary contents.

For more information on the tables and views mentioned in this chapter, see *Data Dictionary*.

Data Dictionary Overview

Data dictionary tables are created when you install the system. The system references some of the Data Dictionary tables directly with SQL requests. Other tables are used only for system or data recovery.

The system also uses Data Dictionary views and macros that access information in the Data Dictionary tables. Views and macros are created by running Database Initialization Program (DIP) scripts.

Data Dictionary Components

The Data Dictionary consists of tables, views, and macros stored in user DBC which:

- store information about all created objects (except volatile tables) in its tables, including, for each object, the:
  - ownership hierarchy (from which parent-child relationship and implicit privileges are derived)
  - explicit privileges
  - type (for example: database, table, or column)

  Use the supplied views to access the Data Dictionary tables.
- automatically update as you and other users create, modify, alter, or drop objects, and grant or revoke privileges.

Display information about a stored Data Dictionary object when you use the SHOW or HELP command.

Fallback Protected

Most Data Dictionary tables are fallback-protected.

Fallback protection means that a copy of every table row is maintained on a different AMP in the configuration. Fallback-protected tables are fully accessible and are automatically recovered by the system.

Non-Hashed

Some Data Dictionary tables contain rows that are *not* distributed using hash maps. Rows in these tables are stored AMP-locally. For example, the TransientJournalTable rows are stored on the same AMP as the row being modified.

User-defined table rows are *always* hash distributed, either with or without a fallback copy. The one exception is a global temporary trace table which is non-hashed.

Unicode Object Name Support

The Data Dictionary defaults to storing object names in Unicode so that the same set of characters are available with the same length restrictions regardless of the character set of a client.

In previous releases, object names with Japanese characters were only guaranteed accessible by a session using the same character set as the session that created the object. However, the Data Dictionary now functions the same whether or not Japanese Language support mode is enabled. Also, object names with Multi-Byte Character Sets can be shared between non-Japanese session character sets.

For more information, see *International Character Set Support* and *Data Dictionary*.

Updating Data Dictionary Tables

Whenever you submit a data definition (DDL) or data control (DCL) statement, the Teradata Database system automatically updates Data Dictionary tables. The following table lists some of the kinds of information recorded by the Data Dictionary.

The Data Dictionary view...	Accesses the Data Dictionary table...	And provides information about...
DBC.AllTempTables	DBC.TempTables	all global temporary tables materialized in the system.
DBC.Children	DBC.Owners	hierarchical relationships.
DBC.Databases	DBC.DBase	databases, users, and their immediate parents.
DBC.Indices	DBC.Indexes	indexes on the tables.
DBC.ShowTblChecks	DBC.TableConstraints	database table constraint information.

The Data Dictionary view...	Accesses the Data Dictionary table...	And provides information about...
DBC.Tables	DBC.TVM	tables, views, macros, triggers, stored procedures, functions, and replication status.
DBC.Triggers	DBC.TriggersTbl	event-driven, specialized actions attached to a single table and stored in the database.
DBC.Users	DBC.DBase	similar to DBC.Database but includes columns specific to users.

For a complete list of all Data Dictionary views, see *Data Dictionary*.

Viewing Updates to Data Dictionary Tables

When you precede your DDL or DCL statement with the EXPLAIN modifier, you can view the updates the system would make to the Data Dictionary tables. This helps you understand what happens when you execute an SQL statement. When you use EXPLAIN:

- The statement is not executed.
- The processing is described, including sorts, use of indexes and spool files, how many response rows are estimated, and so forth.
- The type of locking used, and on what objects, is described. For example:

```
BTEQ -- Enter your DBC/SQL request or BTEQ command:
EXPLAIN CREATE TABLE DBA01.DEPARTMENT (Dept_Number SMALLINT, Dept_Name CHAR(30) NOT
NULL, Budget_Amount DECIMAL (10,2), Manager_Employee_Number INTEGER) UNIQUE
PRIMARY INDEX (Dept_Number);
*** Help information returned. 28 rows.
*** Total elapsed time was 1 second.
Explanation
------------------------------------------------------------------
 1) First, we lock DBA01.DEPARTMENT for exclusive use.
 2) Next, we lock a distinct DBC."pseudo table" for write on a RowHash
    for deadlock prevention, we lock a distinct DBC."pseudo table" for
    write on a RowHash for deadlock prevention, we lock a distinct
    DBC."pseudo table" for write on a RowHash for deadlock prevention,
    and we lock a distinct DBC."pseudo table" for read on a RowHash
    for deadlock prevention.
 3) We lock DBC.DBase for read on a RowHash, we lock DBC.TVM for write
    on a RowHash, we lock DBC.TVFields for write on a RowHash, we lock
    DBC.AccessRights for write on a RowHash, and we lock DBC.Indexes
    for write on a RowHash.
 4) We execute the following steps in parallel.
      1) We do a single-AMP ABORT test from DBC.DBase by way of the
         unique primary index.
      2) We do a single-AMP ABORT test from DBC.TVM by way of the
         unique primary index.
      2) We do a single-AMP ABORT test from DBC.TVM by way of the
         unique primary index.
      3) We do an INSERT into DBC.TVFields (no lock required).
      4) We do an INSERT into DBC.TVFields (no lock required).
      5) We do an INSERT into DBC.TVFields (no lock required).
      6) We do an INSERT into DBC.TVFields (no lock required).
      7) We do an INSERT into DBC.Indexes (no lock required).
      8) We do an INSERT into DBC.TVM (no lock required).
      9) We INSERT default rights to DBC.AccessRights for
         DBA01.DEPARTMENT.
 5) We create the table header.
 6) Finally, we send out an END TRANSACTION step to all AMPs involved
    in processing the request.
 -> No rows are returned to the user as the result of statement 1.
```

For more information on using EXPLAIN, see *SQL Reference: Data Manipulation Statements*.

Dictionary Views and Tables for the Administrator

A BTEQ script called DIPVIEW is typically run during installation and creates the Data Dictionary views. This comprehensive suite of predefined views returns the most critical columns of the dictionary tables. (For complete details of each system view, including column names, values, output format, and underlying system table, see *Data Dictionary*.)

If DIPVIEW has not been run, you can execute the script on an ad hoc basis by running the Database Initialization Program (DIP) utility. For more information, see “DIP Utility” in *Utilities*.

Note: No view is defined for the DBC.IdCol table. This is because DBC.IdCol only has a few columns defined for it. User DBC has SELECT privilege directly on the table. If another user needs to access that table, such as your administrative user, DBC must explicitly submit a GRANT SELECT ON DBC.IdCol to that user.

Most of the dictionary views reference more than one table. They also allow you to limit access to Data Dictionary information and provide a consistent image of the data stored in the Data Dictionary. Most views also record the time of each action in the timestamp.

Because system views …	They help…
• re-title columns • reduce the number of columns returned to those of normal interest • format columns	clarify the dictionary tables and avoid reporting unnecessary details.
• join related dictionary tables • select and project relevant rows and columns	simplify the generation of meaningful responses and reports.
make direct access to the underlying dictionary tables unnecessary	protect the dictionary tables from inadvertent corruption by users.

Administration Views and Tables

You should have granted access to everything in the database hierarchy to your special database administrator user (see “Managing the Database with an Administrative User” on page 125). If you did so, you can access all rows of every view when you log on with your administrator name and password.

Views and tables that might be of particular interest to a database administrator are listed in the following table.

Note: Teradata recommends you use the Unicode versions (the V versions) of the views listed below.

This Data Dictionary object …	Provides information about …
All_RI_Children view	all tables, fully qualified and in referential child-parent order. (It is similar to the RI_Child_Tables view but returns the names instead of internal IDs of the databases, tables, and columns.) The All_RI_Children view is designed for use in a SELECT statement with a WHERE clause to narrow the selection criteria. You can control who has access to internal ID numbers by limiting the access to the RI_Child_Tables view while allowing more (or all) users to access names via this view.
AllRights view	all users and databases who have explicit privileges, and the objects on which the privileges are granted. Included are the name and authority of each granting user, plus an AllnessFlag indicator. • The AllnessFlag, Y or N, indicates whether this privilege was granted to all subordinate users or all users owned by the grantee. • The AccessRight field contains the privilege code; for example, the code AS stands for ABORT SESSION, DG for DROP TRIGGER, PC for CREATE PROCEDURE, and so forth. For more codes, see "Description of Privileges" on page 167.
All_RI_Parents view	all tables, fully qualified and in referential parent-child order. (It is similar to the RI_Parent_Tables view but returns the names instead of internal IDs of databases, tables, and columns.) The All_RI_Parents view is designed for use in a SELECT statement with a WHERE clause to narrow the selection criteria. You can control who has access to internal ID numbers by limiting the access to the RI_Parent_Tables view while allowing more (or all) users to access names via this view.
AllRoleRights view	all rights granted to each role, including the name and authority of the granting user. AllRoleRights is similar to the AllRights view except it does not return the AllnessFlag or CreatorName. For details on creating roles and granting role privileges, see "Using Roles to Manage Privileges" on page 134.
AllTempTables view	all materialized global temporary tables currently materialized in the system. For details, see "Global Temporary and Volatile Tables" on page 55.
AMPUsage view	aggregated CPU and I/O information based on data accumulated in the DBC.Acctg table. Aggregate summaries can be reported by username, accountID, and AMP. Updates to the table are made periodically during each AMP step on each processor affected by the step. For long-running steps, AMPUsage numbers show large increases periodically, instead of continuous incremental additions. In regards to CPU information for co-existence system, the CpuNorm field stores normalized CPU time. In a co-existence systems, AMPs running on faster nodes process data with fewer CPU time than AMPs running on nodes with slower CPUs even though they are processing the same amount of data. CpuNorm is calculated using Teradata-supplied normalization factors to make performance statistics comparable across different hardware platforms.
Children view	hierarchical relationships. The Children view lists the names of databases and users and their owners in the hierarchy.

This Data Dictionary object ...	Provides information about ...
Columns view	table access rights, owners, and the following: • Names, types, attributes, and constraints of all columns in all tables • Column names in views and join indexes • Parameters of macros and stored procedures • Access count and last access time stamps of a table column **Note:** For information on the types and attributes of columns in views, use the HELP COLUMN statement. For the parameters of identity columns, see IdCol table.
Databases view	databases, users, and their immediate owners, the name of the creator of a user or database, and the date and time the user created it. You can also find the access count and last access time stamps of databases. **Note:** Only the immediate owner is identified in this view. Use the parent column of the Children view to select all owners.
DBCInfo view	the attributes of the current Teradata Database software: • Version • Release level
DBQLRules view	the contents of DBQLRuleTbl and returns the query logging rules in effect for a user or account. (For details, see "Chapter 13 Tracking Processing Behavior with the Database Query Log (DBQL)" on page 337.)
IdCol table	parameters defined on identity columns. The next sequential number for new rows of the owning table is the value in the AvailValue field.
IndexConstraints view	any implied index constraints derived from a partitioning expression, including the collation to be used in evaluating the constraint. In the IndexNumber field, a primary index has an index number of 1. For implicit table-level constraints, the value of Name is either the name of the associated index or, if the index does not have a name, NULL. (Also see ShowTblChecks view.)
Indices view	everything that describes a column defined for an index, including the position of that column in the table. Use this view to find the creators of tables with NUPIs that cause skewed, or spiked, row distribution and tables with PPIs that might be causing memory contention. You can also find the access count and last access time stamps of indexes.
LogOnOff view	logon and logoff activity, including attempted but failed logons. The LogonSource column includes OS, TDP, or VM name and more for channel-attached clients. For network-attached clients, the column reports TCP/IP information, port/socket ID, IP address, and more.
MultiColumnStats	statistics collected on columns that are not part of an index.
ProfileInfo view	all profiles and their parameter settings plus comments.
RepGroupTables	the base tables that are members of a replication group.
RepTables view	replication groups defined on the server.

This Data Dictionary object ...	Provides information about ...
RoleInfo view	the names of roles and their creators, plus comments. The column, ExtRole, in the restricted version of this view indicates if the role is a directory-based role.
RoleMembers view	each role name and all of its members.
SessionInfo view	all users who are logged on. **Note:** This view does not include information about session pools, which are collections of sessions logged on under the same logonid. Use the DISPLAY POOL command to find out about session pools (see *Teradata Director Program Reference*). Use this view to find the session source (including host connection, logon name, and application), the current partition, collation, role, password status, and the type of transaction (such as 2PC). For details on how to extract information from this view, see "DBC.SessionInfo View" on page 200.
ShowTblChecks view	each table-level constraint check in the system. Also, see IndexConstraints view. **Note:** Named column-level constraint checks are treated as table-level constraint checks and are reported by this view. Unnamed column-level constraint checks are maintained in the TVFields table.
Tables view	tables, views, macros, join indexes, hash indexes, triggers, stored procedures, user-defined functions, and journals owned by the requesting user or on which he or she has privileges. The following columns simplify searching: • TableKind lets you to limit your response rows to the *kind* of object you are investigating; for example, to see macros only: WHERE TableKind = 'M'. For more codes, see *Data Dictionary*. • PrimaryKeyIndexID identifies the columns used as the primary index, an indicator of whether the NUPI values of a table provide enough uniqueness for even row distribution. • ProtectionType indicates whether the table uses fallback. This is useful if you need to find the fallback condition of every table. For example, to find all NO FALLBACK tables you can use `WHERE ProtectionType='N'`. • The information includes the access count and last access time stamps of the selected tables or other objects.
Triggers view	the definition of every trigger attached to a table.
TVFields table	all views, tables, join indexes, hash indexes, unnamed column-level constraints, stored procedure, and user-defined function parameters. SPParameterType returns stored procedure parameters such as IN, INOUT, or OUT. IdColType returns information about an IDENTITY column.

This Data Dictionary object ...	Provides information about ...
TVM table	every table, view, trigger, hash indexes, join index, macro, stored procedure, and user-defined function in the system. If a table is defined as a replication group, the TVM table also contains the state of the table, the replication group identifier, and the table role. The possible values for table role are null, C if a capture operation is active, and A if an apply operation is active. The value in the SPObjectCodeRows field indicates stored procedure creation-time attributes, including: • Session mode • Server platform • Print option • SP text storage option • Teradata Stored Procedure (TDSP) version number
Users view	users that the requesting user owns or has MODIFY or DROP privileges on. Because this makes it a user-restricted view, there is no X version. (However, if you select this view when logged on as user DBC, all current users are returned because user DBC owns everything.) The information returned includes: • Date and time a user is denied access due to excessive erroneous passwords • The number of failed attempts since the last successful logon • Creator name and timestamp • Last altering user name and time stamp

Note: The output of these views extends beyond 80 characters. To capture all the columns and their contents, you can either cast the name columns to shorter lengths or use the BTEQ .SET FOLDLINE command.

User-Restricted Views

System view names followed by an X appended to the viewname indicate a user-restricted view. This means that the contents shown from the view apply only to the user submitting the query that acts upon a viewname.

View Type	Description
Standard	Reports information associated with all applicable users.
X	Reports only a subset of the available information. These views are defined with a `WHERE User=`*username* clause, which returns information only on objects the requesting user owns or on which the user has privileges. These views are also called limited or restricted views. The limited views are identified by the character X, which is appended to the view name (for example, TableSizeX).

Note: Queries on X views cause verification of user privileges on the underlying objects and return only the relevant rows. Thus, X views may take longer to respond than standard views. However, queries on standard views usually return many more rows.

In some cases, separate viewnames are defined for restricted and non-restricted views; for example, UserRoleRights and AllRoleRights.

DiskSpaceX, TableSizeX, and SessionInfoX are the names of restricted views[1] that show only information for objects owned or are accessible by the submitting user; data from DiskSpace, TableSize, and SessionInfo is information about all users in the system.

Returns from Non-Restricted Views

An unqualified select on a non-restricted view returns all rows from the underlying tables, which can overflow user spool space. Also, unless you explicitly revoke access to it, the view lets any user access all the information.

Returns from Restricted Views

X views have the same columns as non-X views. One difference, however, is that the definition includes a WHERE clause which limits access on the underlying tables to only those rows associated with the requesting user. The user can only view objects he or she owns, is associated with, has been granted privileges on, or is assigned a role which has privileges. For example, if UserA submits:

```
SELECT * FROM DBC.ProfileInfoX ;
```

the response is the name and parameter settings only for the profile assigned to UserA.

This makes the response meaningful, limits its row size, and protects user privacy.

Restricted views typically run three different tests before returning information from Data Dictionary tables to a user. Each test focuses on the user and his or her current privileges.

If the current role of the user is not NULL or ALL, then the system checks for privileges granted to the current role and its nested roles before returning information from the restricted views. If the current role of the user is ALL, then the system checks privileges granted to all the roles to which he or she is directly granted and to all their nested roles. Consequently, it may take longer to receive a response when selecting from a qualified restricted view.

Restricting PUBLIC Access to Views

By default, the SELECT privilege is granted to PUBLIC (and is therefore available to all users) on most views in both the restricted and non-restricted versions.

Some views are applicable only to users who have a need to see specialized information, such as a database administrator, security administrator, or Teradata field service representative. Access to these views should be limited to only applicable users. For example, you should grant privileges to the DBC.DBQLRules view only to appropriate users or roles.

You can revoke from PUBLIC to remove one or more privileges so that those privileges are no longer available via PUBLIC to all users. Also, you can use GRANT and REVOKE to give or take away one or more privileges on any view for a particular user or role.

For more information on GRANT and REVOKE see *SQL Reference: Data Definition Statements*. For more information on individual views, see *Data Dictionary*.

[1] The V suffix indicates a Unicode version of the view. For more information, see *Data Dictionary*.

Compatibility Views Versus New Unicode Views

The Data Dictionary contains Unicode views. However, to maintain backwards compatibility, Teradata provides Compatibility Views to translate the object name to Kanji1/Latin.

New Unicode Views (or V Views), on the other hand, pass the Unicode string and return the same answer set independent of the language support. A Unicode view exists for each system view in database DBC. This means that a Unicode View is added even if the old (existing) view does not access object name columns from the dictionary tables. Teradata recommends you use the Unicode version of views.

All Unicode system views end with a suffix V (or VX if you are using restricted Unicode views). If you add any new views to database DBC, add a V suffix in the view name. For more information, see “Unicode System Views” in *Data Dictionary*.

Maintaining Data Dictionary Tables and Logs

Managing the size of Data Dictionary tables (sometimes called logs) can help improve performance. Teradata 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. (With respect to keeping data dictionary data for historical purposes or trend analysis, archive tables using an OTB solution.)

Purging the System Logs

The system does not automatically purge logs or the tables underlying views such as the AccessLog, LogOnOff, and Software_Event_Log views. You or another authorized user, such as the security administrator, should archive (as desired) and then delete information that is older than 60 to 90 days (or some interval that suits you) from the following:

- DBC.ResUsage
- DBC.AccLogTbl
- DBQL logs, which include all the DBQL tables.

  (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 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 *SQL Reference: Statement and Transaction Processing* and "COLLECT DEMOGRAPHICS" in *SQL Reference: Data Definition Statements*.

In addition to maintaining the size of system tables and logs, you can proactively reduce their size as follows:

- Using only DBC.ResUsageSpma instead of activating logging on multiple ResUsage tables. ResUsageSpma may be able to provide all the information you need.
- Using roles to reduce the size of DBC.AccessRights. For more information on how to create roles, see *SQL Reference: Data Definition Statements*.
- Tracking only the information you need for ResUsage, DBQL, or accounting.

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 "Tracking Accounts With Account String Expansion (ASE)" on page 148.

For more discussion, see "Cleaning Out Frequently Updated Tables" on page 299.

Using Teradata Manager

Use the Data Collection tab of the Administrate menu in Teradata Manager to specify the number of days you wish to keep certain the detail data (Retention Period for Detail Data). This includes:

- AMP usage data
- DBQL data
- Teradata DWM data
- Priority Scheduler data
- Query band pairs data
- Resource usage data

Then schedule Teradata Manager to run the Cleanup function or reset peak spool space statistics.

Note: Do not drop these system tables. You should only delete the entries in them when the data in the tables are no longer needed. Remember, you can always archive the information in the tables if you want to review them at a later time.

You can also manually submit SQL statements to delete old data from these tables or reset values.

Clearing DBC.Acctg Table with DBC.AMPUsage View

The DBC.AMPUsage view uses the DBC.Acctg table to provide aggregated information about CPU time and disk usage. 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 "Tracking Accounts With Account String Expansion (ASE)" on page 148). However, use caution when purging rows containing accounts with ASE codes.

Caution: 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.

You can use the following process to first save the data, then clear out DBC.Acctg and reset the counters:

1. To control growth, at relatively short intervals use Teradata Manager to:

   a. Summarize DBC.AMPUsage data.

   b. Save the summaries to a history table.

   c. Delete the detail rows from DBC.Acctg.

   For instructions, see *Teradata Manager User Guide.*

2. At longer intervals, archive the entire predefined set of system tables.

3. After the accumulated AMP data is successfully summarized and archived if necessary, reset the CPUTime and DiskIO counters to zero with the following:

   ```
   UPDATE DBC.AMPUsage SET CPUTime = 0, DiskIO = 0 ALL ;
   ```

If you use ASE but are not sure which rows to purge from the DBC.Acctg, do the following:

- Create a table similar to DBC.Acctg with a non-unique primary index, date, and time columns.
- Populate the table with rows which are created for each respective account by capturing the sum total of the relevant accounting fields at regular intervals.
- 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 space values in DBC.DatabaseSpace	DBC.ClearPeakDisk macro	"Resetting Peak Values" on page 105.
accumulators in DBC.Acctg	the statement: `UPDATE DBC.AMPUsage SET CPUTime=0, DiskIO=0 ALL;` **Note:** If using ASE, construct a WHERE clause to avoid deleting ASE information.	"Clearing DBC.Acctg Table with DBC.AMPUsage View" on page 121.

IF you want to reset ...	THEN use ...	For instructions and examples, see ...
count values, AccessCount and LastAccessTimeStamp fields, in: • DBC.Dbase • DBC.TVM • DBC.TVFields • DBC.Indexes	one of the DIPVIEW macros: • ClearAllDatabaseUseCount • ClearDatabaseUseCount • ClearTVMUseCount	"Resetting the Use Count Fields" on page 332.

Correcting DBC.DataBasesSpace 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.DataBasesSpace and DBC.DBase. Use the Update DBC and Update Space utilities only when you need to perform these tasks.

<table>
<tr><th>Utility</th><th>Description</th></tr>
<tr><td>Update DBC</td><td>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.DataBasesSpace system table.</td></tr>
<tr><td>Update Space</td><td>Examines the storage descriptors and recalculates the current usage of space used by:
• A single database and its individual tables
• All databases in a system and their individual tables
Then Update Space utility updates the CurrentPermSpace, CurrentTempSpace, and CurrentSpoolSpace columns in the DBC.DataBasesSpace table for each table and for the containing database as a whole.
<table>
<tr><th>The Update Space utility recalculates...</th><th>In the system table ...</th></tr>
<tr><td>MaxPermSpace, MaxSpoolSpace, and MaxTempSpace values for each database in the system based on the PermSpace, SpoolSpace, and TempSpace values in DBC.DBase for that database</td><td>DBC.DataBasesSpace (AMP)</td></tr>
<tr><td>PermSpace, SpoolSpace, and TempSpace values:
• The PermSpace value in DBC.DBase for user DBC is the total available storage space minus the PermSpace for all other databases.
• The SpoolSpace and TempSpace values in DBC.DBase for user DBC are the total available storage space.
For databases other than DBC, the PermSpace, SpoolSpace, and TempSpace values in the DBC.Dase table are the maximums declared when the database is defined.</td><td>DBC.DBase (global)
<b>Note:</b> Values in DBC.DBase are global values. Values in DBC.DataBasesSpace are local AMP values, that is, the global value divided by the number of AMPs in the system.</td></tr>
</table>
</td></tr>
</table>

For instructions on each utility, see *Utilities*. For descriptions of the system tables and the views that access them, see *Data Dictionary*.

CHAPTER 5 Setting Up Users, Roles, Profiles, and Accounts

This chapter discusses best practices for creating an administrative user to manage the system. Other topics include considerations for creating users, roles, profiles, and accounts.

Best Practices for Setting Up Users and Roles

While there is no one 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.

- Create *separate* users for the security administrator and database administrator.
- Ensure that all users are uniquely identified. This means not allowing several users to log in to Teradata Database using the same username.
- Grant only the minimum number of privileges to users required for their job.

  Do not grant an administrative user all the same privileges as user DBC. Only grant the privileges required for the administrative user to do required tasks. The same principle applies to other users. Granting a user more access than he needs causes unnecessary privilege checks as well as puts the user at risk of unintentionally modifying or deleting objects. No user should have access to something for which they do not require access.
- Use roles to control and manage privileges according to job description or responsibility.

  Using roles to manage privileges simplifies the granting and revoking of privileges and helps maintain the size of the DBC.AccessRights table.

Managing the Database with an Administrative User

Teradata recommends that you create an administrative user from User DBC and log on as that user to perform normal daily administrative tasks. Teradata also recommends creating a separate security administrator to perform security-related tasks. (See “Typical Privileges Granted to a Security Administrator” in *Security Administration*.)

Creating an administrative user from User DBC is standard practice and allows you to protect sensitive data and system objects owned by User DBC. It enables you to manage all the users and databases you subsequently create.

You can name the user anything except “sysadmin” or any other name already reserved by the system. This section calls the user DBADMIN. When you assign space to DBADMIN, remember to leave enough space in DBC to accommodate the growth of system tables, logs, and the transient journal.

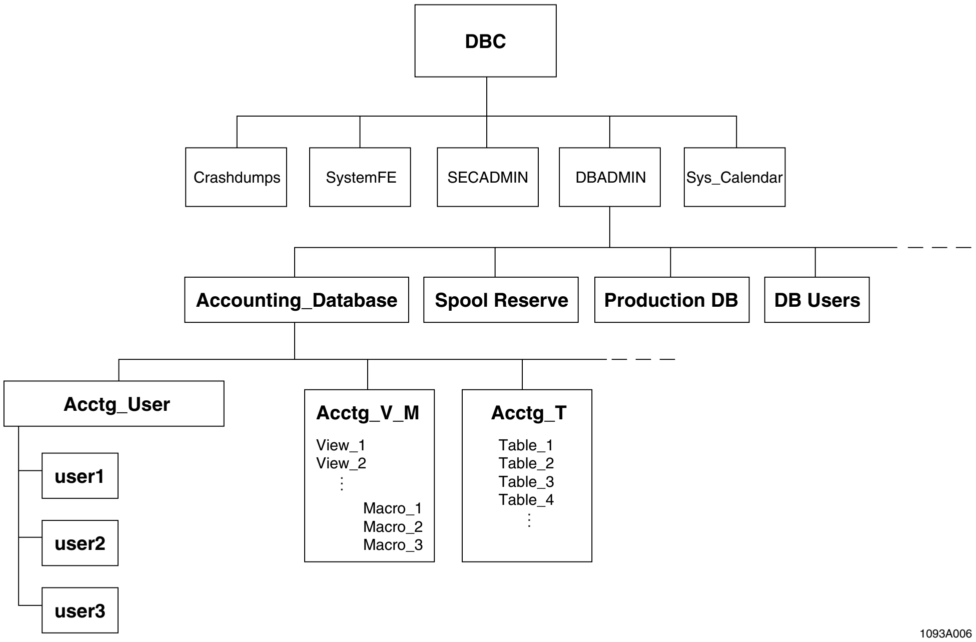
First estimate the following:

- The maximum-sized Transient Journal (see “Determining Maximum TJ Size” on page 231)
- Growth of the DBC.AccessRights system table
- The maximum size of DBC.DBQLogTbl, based on your planned table maintenance if you implement Database Query Logging (DBQL) (see “Purging the System Logs” on page 120 and “Main Log Table: DBQLogTbl” on page 348)
- The growth of DBC.Acctg, based on planned table maintenance (see “Clearing DBC.Acctg Table with DBC.AMPUsage View” on page 121)
- The growth of logs based on how many users, actions, and objects you track if you plan to use security access logging
- The amount of space to be set aside for DBC (“Reserving Minimum Spool Space” on page 97)

Then, assign as much of the remaining perm space as possible to your database administrator and security administrator according on your site requirements and policies.

After using the GRANT...WITH OPTION statement to grant all required privileges retained by DBC to your new DBADMIN user (or whatever name you chose), log on as DBADMIN to create all your Teradata Database users and high-level databases. This ensures that DBADMIN owns each user and high-level database (and thus each will be a child of DBADMIN) and also the objects created in them.

Teradata recommends that you set up users and databases similar to the following hierarchy.



This hierarchy helps protect data and ensures users do not inadvertently make changes to the actual data in the base tables. For example, users in the accounting department do not have direct access to any of the accounting tables. Instead, they are assigned to a role called "Acctg_User" created for accessing accounting information through views and macros.

If you have multiple people acting as database or security administrators, consider creating a role for DBAs, a role for security administrators, and a role for shared administration duties. Also refer to "Establishing Administrative Users" in *Security Administration*.

Creating Users

Consider the following things for creating new users:

- Based on job function, determine which roles or profiles to assign the user.
- Determine implications on available permanent, temporary, and spool space. (For more information, see Chapter 3: "Space Considerations.")
- Resolve ownership issues.
- Define authorization checks and validation procedures.
- Audit LOGON, GRANT, REVOKE, session, and other account and access-related activity.
- If using corporate LDAP-based authentication and authorization infrastructures to centrally manage users, carefully map directory users with database-based users. Map logons from external applications that run reports or queries against the database to the system-provided EXTUSER. (See "Directory Users" on page 130.)

Creating New Users

The CREATE USER statement enables you to add new users to the system. In addition to a unique username, CREATE USER requires that you define the following:

- A password. You must supply a password for a user in the CREATE USER statement.
- Permanent space. Permanent space comes from, and cannot exceed, the current PERM of the immediate owner, which is either your default database, or the database you specify in the FROM clause of the CREATE USER statement.

You can optionally define defaults:

- Default database. During a session, each user is associated with a default database, which is the space in which the Teradata Database stores or searches for new or target objects unless a different database is specified in the transaction. By default, the default database has the same name as the username. You can define a different default database with:
  - The DEFAULT DATABASE clause of the CREATE USER statement.
  - A profile, which takes precedence over the CREATE USER definition.
  - During a session, the SET SESSION DATABASE statement, which takes precedence over a profile or a CREATE USER definition.
- Default role. To use a role, the username under which you logged on must have been explicitly granted that role, or must have an active role (current role or a role nested within

it) that has been granted that role. In addition, the GRANT statement must include the WITH ADMIN OPTION. (See "Rules for Using Roles" on page 136.)

Use the CREATE USER statement to define a user account string, temporary space, spool space, or additional password attributes. Or, use profiles to more easily manage these parameters at the group level. (To assign a profile to a user, you need the DROP PROFILE privilege. The definitions in a profile override any definitions in a CREATE/MODIFY USER statement. For information on default settings for profiles, see "Profile User Defaults" on page 142.) For more information, see CREATE USER and CREATE PROFILE in *SQL Reference: Data Definition Statements*.

CREATE USER Default Values

The following table summarizes default values associated with the CREATE USER statement.

Note: PERM space and PASSWORD clauses have no default values; both must be specified at user creation time.

<table>
<tr><th>IF the DDL does not define ...</th><th>THEN the default value used is ...</th></tr>
<tr><td>FROM <i>databasename</i></td><td>the default database of the creating user.</td></tr>
<tr><td>SPOOL</td><td>one of the following:</td></tr>
<tr><td></td><td>
<table>
<tr><th>IF a profile is...</th><th>THEN the value used is the...</th></tr>
<tr><td>assigned to this user that has a SPOOL value</td><td>limit defined in the profile.</td></tr>
<tr><td>assigned but does not have a SPOOL value</td><td>same SPOOL value as the immediate owner of the space in which the user is being created.</td></tr>
<tr><td>not assigned</td><td>same SPOOL value as the immediate owner of the space in which the user is being created.</td></tr>
</table>
</td></tr>
<tr><td>TEMPORARY</td><td>one of the following:
<table>
<tr><th>IF a profile is...</th><th>THEN the default is the...</th></tr>
<tr><td>assigned to this user and it has a TEMPORARY value</td><td>limit defined in the profile.</td></tr>
<tr><td>assigned but does not have a TEMPORARY value</td><td>same TEMPORARY value as the immediate owner of the space in which the user is being created.</td></tr>
<tr><td>not assigned</td><td>same TEMPORARY value as the immediate owner of the space in which the user is being created.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>IF the DDL does not define ...</th><th>THEN the default value used is ...</th></tr>
<tr><td>STARTUP</td><td>null (no startup string). The user can enter a startup string during logon.</td></tr>
<tr><td>DEFAULT DATABASE</td><td>the username. The user can submit a SET SESSION DATABASE statement to change the default for the current session, as long as the appropriate access rights on that database have been granted to the user.</td></tr>
<tr><td>DEFAULT ROLE</td><td>null. No role is used in the privileges rights validation process unless the user submits a SET ROLE statement. The user can submit a SET ROLE statement only if that role has been explicitly granted to the user.
You can use SET ROLE ALL so that the system validates privileges from every role you are assigned, directly-granted and nested, within the current session. For more information, see “Rules for Using Roles” on page 136.</td></tr>
<tr><td>ACCOUNT</td><td>one of the following:
<table>
<tr><th>IF a profile is...</th><th>THEN the default is the...</th></tr>
<tr><td>assigned to a user but the profile contains more than one account</td><td>first account in the string. The user can submit a SET SESSION ACCOUNT statement to set the default for the current session to one of the other accounts.</td></tr>
<tr><td>assigned to a user and the profile contains an account</td><td>account defined in the profile.</td></tr>
<tr><td>assigned to a user but the profile does not have an account assigned to it</td><td>account identifier of the immediate owner of the user.</td></tr>
<tr><td>not assigned to a user</td><td>account identifier of the immediate owner of the user.</td></tr>
</table>
<b>Note:</b> If a user <i>does</i> have an account defined to it but not a profile, then the default account is the account string of the user itself.
If you assign a profile to a user and that profile has an account string defined, then the profile account will be in effect for AMPUsage and DBQL for that user. However, the value in DBC.DBase (accessible through the DBC.Database view) will always be used for space accounting.</td></tr>
</table>

Granting Create Privileges to a New User

Certain explicit privileges are granted automatically when CREATE USER is processed successfully:

- The privileges of a newly created user are granted on the user space, enabling the creation of tables, views, and other data objects.
- Creator privileges are granted to the creator of a newly created user, database, or object.

In addition, all owners of the space from which the user was created have implicit privileges on the space.

A new user has the privilege to create data objects (such as tables, indexes, permanent journal tables, views, macros, triggers, and so on) in his or her default database, as long as the new user has the appropriate privileges on any underlying tables and target tables created by another user or residing in another database.

This means that new users can immediately create tables and other data objects (such as views, macros, indexes, and triggers) in their own space. However, there are 12 privileges that you must explicitly grant to a new user or database on itself. For a list of these privileges, see "Privileges That Must Be Explicitly Granted" on page 165.

When granting privileges, the grantor does not need to have any privileges (or WITH ADMIN OPTION) on the grantee role, user, or database in order to grant it a privilege. For example, a user without any privilege on a role can grant privileges to that role. However, the grantor does need privileges (WITH GRANT OPTION) on the objects that are being granted.

For more information on:

- Which privileges are automatically granted by the system and which must be explicitly granted, see "Privileges Automatically Received" on page 163 and "Privileges That Must Be Explicitly Granted" on page 165.
- Creation, assignment, and default values of roles and profiles, see "Using Roles to Manage Privileges" on page 134.
- MODIFY USER statement and the privileges required to change the options of an existing user, see *SQL Reference: Data Definition Statements*. To use MODIFY USER, you must have DROP USER privileges.

Directory Users

A directory user is defined on a directory server instead of on Teradata Database itself. Setting up directory user access to Teradata Database involves mapping the user in the directory entry to either a permanent user in Teradata Database or to a Teradata supplied generic user called EXTUSER (EXTernal USER). The actual creation of a directory server entry must be coordinated between the DBA and the directory server administrator.

Directory users are required to have at least one mapping to a Teradata user, profile, or role in order to log onto the database. For more information, see "Directory Server Authentication" on page 191.

Directory users mapped to database-based users can create users and databases if permitted by the privileges granted to them. The users and databases they create are registered in DBC.Dbase with the mapped permanent users as the owners and creators. For more information on mapping directory-based users to database-based users, see "Directory Users" on page 130.

Note: Teradata supports two Lightweight Directory Access Protocol (LDAP) version 3 compliant directory servers: Microsoft Active Directory and Sun Java System Directory Server. For information on how to set up a directory for integration with your Teradata Database, refer to "Integrating an External Security Directory" in *Security Administration*. You should also refer to the documentation for your specific directory. For more information on open source LDAP, go to http://www.openldap.org.

Directory users and EXTUSER have the following characteristics on Teradata Database:

- If a directory user is not mapped to a permanent user, and the directory user happens to have the same name as a defined permanent user, logon will fail. However, if the directory user is mapped to a permanent user, even if they have the same name, the directory user can successfully log in.
- If an external user is mapped to a role that does not allow the create or drop statement for users, profiles, and other roles, EXTUSER also has the same restriction.
- EXTUSER has no perm space, default database, default role, or profile. External users can only create database objects in another database if permitted by roles and profiles assigned to them.
- An external user mapped to EXTUSER will have the same default spool and temp space limits as a permanent user that is created without spool and temp space specifications. A profile assigned in the directory may be used to cap the spool and temp space usage of a user.
- An external user mapped to a permanent user will assume the same perm and spool space as the permanent user stored in DBC.Dbase. A directory-assigned profile may be used to cap the spool and temp space usage of an external user.
- An external user mapped to both a permanent user as well as to EXTUSER will default to using the mapping to the permanent user.
- Though there is an entry in DBC.Dbase for EXTUSER, it is not a valid user or database that can be referenced in SQL statements.
- The privileges for EXTUSER comes from external roles assigned to the users in the directory plus PUBLIC rights.

The LDAP column of DBC.SessionTbl denotes whether a given session has any name mapping involved. The LDAP column reports “P” for sessions of an external user mapped to a permanent user, “X” for sessions of an external user not mapped to any permanent user, and “N” for sessions that are not directory-based.

The Username column of DBC.SessionTbl denotes the username of a given session. For a directory user, the username corresponds to the authentication identifier (authcid) specified in the directory server entry.

For examples and more information on creating external users, see “Directory-Based Management of Database Users” in *Security Administration*.

Password Security for Users

The Teradata Database 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.

While you cannot create a user definition with a null password, you can permit an existing user to log on with a null password if your site employs external authentication. External authentication allows network users to access authorized resources, applications, and the data warehouse, with a single authentication that is performed during the initial network access. To use external authentication, be sure to use the proper security settings as described in "Single Sign-On" on page 188.

The following table describes some security considerations for your password policy.

IF your site ...	THEN ...
requires use of passwords	Create each user in the system with a password. Specify the following: • A reasonable time for password expiration and rules for password reuse • The maximum allowable logon attempts and duration of user lockout • Maximum or minimum number of characters • Whether or not special or numeric characters are allowed You can optionally require passwords to: • Contain mixed case • Have at minimum one alpha character • Have at minimum one numeric character • Have at minimum one special character • Not contain the username as part of the password string or prohibit words from the DBC.PasswordRestrictions table from being used in the password. Specify these password rules through the DBC.SecurityDefaults views. For more information, see "Customizing Your Password Controls" on page 186. To simplify the management of the required and optional password attributes, use profiles to specify settings for groups of users. Then you can simply assign users specific profiles so that they will inherit those password requirements and restrictions. For more information, see "Using Profiles to Manage Settings" on page 140.

determines that using null passwords is permissible according to security policies	Follow this procedure:
	1 Log on as the user with EXECUTE privilege on DBC.LogonRule macro.
	2 Create each new user with a temporary password.
	3 Follow the CREATE USER statement with a GRANT LOGON…WITH NULL PASSWORD statement for that username. For example:
	`CREATE USER JANE AS PERM=1000000, PASSWORD=Jane;`
	`GRANT LOGON ON ALL TO JANE WITH NULL PASSWORD;`
	Note: You need special privileges to use the GRANT LOGON statement. Teradata suggests that you create a special user to serve as your security administrator (for example, SecAdmin) and grant the EXECUTE privilege to that user on the special security macro DBC.LogonRule.
	4 For channel-connected mainframes, write a TDP security exit to register that the logon string for this username is valid without a password. (For more details, see *Teradata Director Program Reference*.)
	A null password applies only to logging onto Teradata Database; other security measures still apply. For cautions and instructions on using null passwords, see *Security Administration*.
	Note: Under any circumstance, a null password limits the ability of Teradata Database to authenticate the identity of a user.

IF your site ...	THEN ...

is using LDAP external authentication with implied logons	Follow each CREATE USER statement with:
	`GRANT LOGON ON ALL TO username WITH NULL PASSWORD;`
	If you are using NTLM, KRB5, or LDAP where the LDAP directory is Active Directory and the server is running on Windows, you may need to append domain names to make every logon name unique across all domains[a]. To do this, perform the following procedure:
	1 Query the Append Domain Name value with the -d option of the gtwcontrol utility. This value determines what form of username is accepted and the default is “off”.
	When Append Domain is set to off, *username* is the only form accepted. If Append Domain is set to yes, only the format "*username@domainname*" is accepted.
	To change the current value, toggle it by entering the -F option to the gtwcontrol command:
	`gtwcontrol -F`
	However, note that the -F flag is deprecated in Teradata Database 12.0 because the value will be permanently set to off in future releases.
	2 If the system accepts domain names, append a domain name to the user name. To do this, define each username in the form:
	`"username@domainname"`
	For example, to create user Bob for domain esdev3, enter:
	`CREATE USER  "Bob@esdev3" AS PERM=10000000, PASSWORD=Bob;`
	`GRANT LOGON ON ALL TO "Bob@esdev3" WITH NULL PASSWORD;`
	Make sure the DBS Control field “External Authentication” is set to 0 (the default which is to accept both external authentication and traditional logons) and the Gateway Control -a externalauthentication option is also set to ON.
	The -b AllowDeprecatedLogons setting of Gateway Control should be set to no. However, note that this flag is also deprecated in Teradata Database 12.0 and will be permanently set to no in future releases.
	For step-by-step instructions on the complete procedure, see “Single Sign-On” on page 188.

[a] In future releases, Teradata will no longer support non-unique user names.

Note: Granting access with NULL passwords does not imply that the user is allowed to logon without a password. The user is simply authenticated externally from the database upon initial network access if you set up single sign-on.

Creating Temporary Passwords for First Login

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

- At the time of installing a new system, he or she 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.

If you want to create a temporary password and prompt users for a new password upon login, use the FOR USER phrase of the PASSWORD option in the MODIFY USER statement. You will also need to change the expire value of the profile for the user or change the value of ExpirePassword field in DBC.SecurityDefaults to a value greater than 0.

Caution: You must restart the DBS for the changes to apply. Changing the ExpirePassword value affects ***all*** users who subsequently log in. Be sure to set ExpirePassword to 0 again after users have changed their passwords.

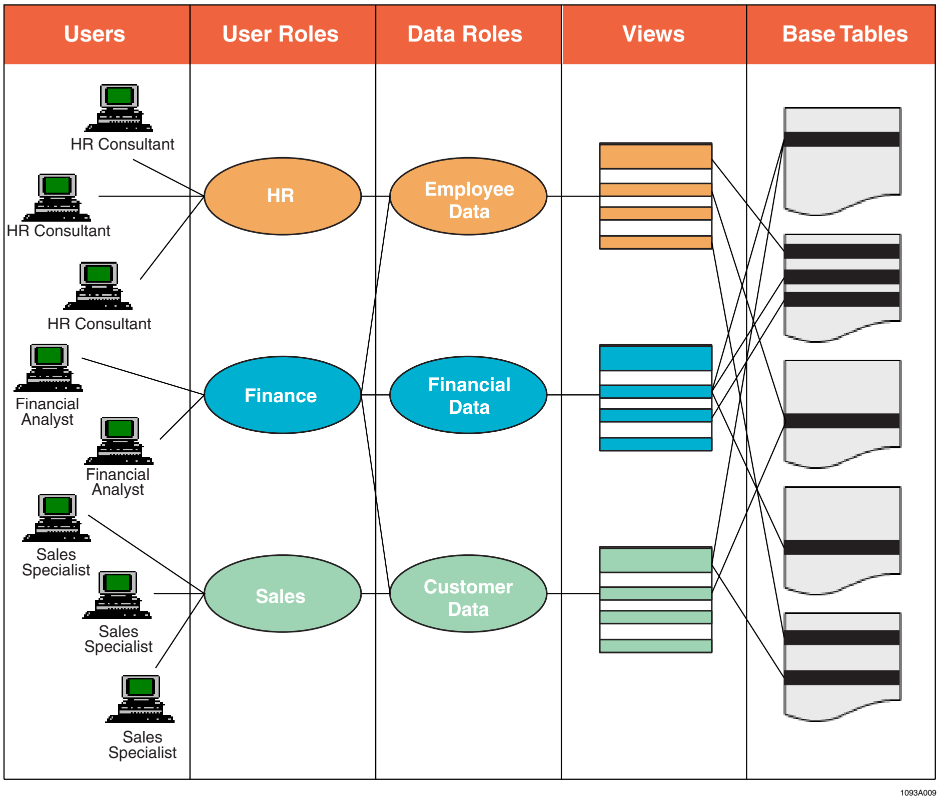
If you are using Teradata Administrator, simply check the “Temporary” box next to the password. This adds the 'FOR USER' phrase to the generated command.

To use SQL, see “Procedure for Creating a Temporary Password” in *SQL Reference: Data Definition Statements*.

Using Roles to Manage Privileges

Simplify the management of privileges for users with roles. Roles help improve performance by reducing the number of rows added to and deleted from DBC.AccessRights. To use roles, assign privileges to a specific role and then grant the role to users. The users can then access all the objects on which their assigned role and nested roles have privileges.

The following diagram shows users assigned to roles which group them by function. These roles in turn are assigned to nested roles which have the privileges to access only specific views. These views only access the data they need in the base tables.



To manage privileges using roles, do the following:

- Decide which privileges to which objects a role should have. Roles work best when based on job functions or responsibilities. Some users may belong to more than one role.
- Create different roles for the different job functions and responsibilities.
- Grant specific privileges on database objects to the roles.
- Grant the role to users or other roles. However, note that roles can only be nested one level.
- Assign default roles to users.

Assigning a default role to a user allows the user to automatically access all the objects to which the role has been granted privileges. It also gives the user additional privileges to all the objects its nested roles also owns.

A newly created role does not have any associated privileges until they are granted. Any user except user DBC must be explicitly granted the CREATE ROLE privileges in order to create, drop, grant, and implement a role (see the following section "Creating Roles").

When using roles with directories, the directory administrator assigns external roles to directory users. External roles are essentially the same as internal roles. The only difference lies in how they are assigned to users. For example, external roles cannot be assigned to users or other roles via the SQL GRANT ROLE statement. "Rules for Using Roles from a Directory (External Roles)" on page 137 lists some guidelines for using roles from a directory. For more detailed information on integrating directories with your Teradata Database system, see *Security Administration*.

Creating Roles

When you create a role, you automatically receive creator privileges (implicit privileges via ownership do not apply). This gives you the DROP ROLE privilege and the WITH ADMIN OPTION privilege.

Note: Creator privileges do not give you the privilege to assign a default role to a user. These parameters are specified in the DEFAULT ROLE clause of CREATE/MODIFY USER, for which you need the CREATE USER or DROP USER privilege, respectively.

As long as you have the WITH ADMIN OPTION, creator privileges enable you to do the following:

- Drop any role you create.
- Grant any role you create to other users and roles.
- Grant a role you create to another user with the WITH ADMIN OPTION, enabling that user in turn to grant, revoke, or drop that role to other users.
- Revoke any role you granted (including the WITH ADMIN OPTION privilege, if you granted it along with the role).

If a directory-based user is mapped to a database-defined user, then the method for creating roles is the same. However, permission to assign a role is based upon directory access control entries rather than the possession of the WITH ADMIN option over the role. Also, roles assigned in the directory will not have a corresponding row inserted in DBC.RoleGrants.

Rules for Using Roles

When a user who is defined in Teradata Database logs on to the system, the assigned default role is the initial current role for the session. This current role is used to authorize privileges after all checks against individually granted privileges have failed. Once the session is active, the user can submit a SET ROLE statement to change or nullify the current role. (For example, if a user is assigned to RoleA and RoleB, but logs in as RoleA, then the system checks against RoleA and all nested roles for privileges. The user cannot use the privileges of Role B. To use privileges of both RoleA and Role B, the user must activate all roles with the SET ROLE ALL statement.)

The rules for using roles are as follows:

- You can grant one or more roles to one or more users or roles; thus,
  - A role can have many members
  - A user or role can be a member of more than one role
- Only single-level nesting is allowed; that is, a role that has a member role cannot also be a member of another role.
- A privilege granted to an existing role immediately affects any user or role that is specified as a recipient in the GRANT statement and currently active in a session.
- The privileges of a role granted to another role are inherited by every user member of the grantee role.
- Using SET ROLE ALL allows all roles available to a user to be enabled within a session. Available roles are those that have been directly granted to the user as well as those that are nested within the granted roles. All available roles may be enabled by one of these methods:
  - dynamically by submitting a SET ROLE ALL statement or
  - upon logon if the default role of a user was set to 'ALL' through a CREATE USER or MODIFY USER statement.
- Users may set their current session to ALL even if the users have no roles granted to them. No privilege is required for users to do so.
- When using CREATE USER…AS DEFAULT ROLE ALL, the creator is not required to be granted any roles.
- When using MODIFY USER…AS DEFAULT ROLE ALL, the user is not required to be granted any roles.

  When you grant a role to a user with the WITH ADMIN OPTION, the grantee is enabled to do the following:
  - Drop the role
  - Grant the role to other users and roles
  - Grant the role to another user with the WITH ADMIN OPTION
  - Revoke the role from a grantee

Rules for Using Roles from a Directory (External Roles)

If you are logging on as a directory-based user, and the proper set up has been made on the directory server, you can have all directory-assigned roles enabled within a session. The initial current roles for the session are any external roles mapped to the directory user. If there are no mapped external roles, then the directory user will use Teradata user mappings.

If a directory user who is mapped to a permanent database user logs on with external roles assigned, the external roles override the default role of the permanent user as the initial current role for the session. The initial set of enabled roles does not include database-assigned roles. If, however, no external role is assigned, then the default role of the permanent user and its nested roles are the initial set of enabled roles. If neither external nor database-assigned roles are assigned to the user, then the initial current role is NULL.

The following list describes some usage considerations for directory-based roles:

- Unlike database-assigned roles, the creator of an external role is not implicitly granted creator privileges to the role. This means that the system does not insert a row into the DBC.RoleGrants table when the role is created.
- To create or drop roles that are assigned in the directory, use the CREATE EXTERNAL ROLE or DROP EXTERNAL ROLE statements. For more information on the SQL syntax, see *SQL Reference: Data Definition Statements*.
- You must have the CREATE ROLE and DROP ROLE privileges to create and drop external roles.
- Use SET ROLE `rolename` to change the current role or SET ROLE ALL to enable all external and internal roles available to the user within a session. The SET ROLE request does not distinguish between directory and database-assigned roles.
- External roles may not be granted with the GRANT statement. Only individual rights and database roles may be granted to external roles. For example, you can submit the following: `GRANT CREATE DATABASE ON example_db TO externalrole01;`

  If a user mapped to EXTUSER uses that privilege in externalrole01 to create a database, the owner of the newly created database will be example_db.
- If an external role is dropped when directory users are logged on, then all users logged on with that role will immediately cease to have the privileges granted through that role. The same occurs with database-assigned roles.

  For more information on the SQL required to create, manage, and drop external roles, see *SQL Reference: Data Definition Statements*.

The following user...	may have access to external roles...	may have access to internal Roles...
EXTUSER	Yes	No
Directory user mapped to a Teradata Database user	Yes	Yes
Teradata Database user	No	Yes

Examples of Using Roles

Assume you have created a database administration user named DBADMIN. Also assume that user DBC has already granted every privilege on the parent of the Accounting database to DBADMIN using WITH GRANT OPTION.

DBADMIN submits the following statements to implement roles for the Accounting database:

```
CREATE ROLE Role1;
GRANT SELECT ON Accounting TO Role1;

CREATE ROLE Role2;
GRANT SELECT, UPDATE, INSERT, DELETE ON Accounting.AccPay TO Role2;

GRANT CREATE ROLE TO Charles;
GRANT Role1 TO Alan, Betty, David, Ellen;
GRANT Role1, Role2 to Charles WITH ADMIN OPTION;
```

Charles can now create new roles and submit any of the following commands:

```
CREATE ROLE Role3;
GRANT Role1, Role3 TO Francis;
REVOKE Role1 FROM David;
DROP ROLE Role2;
```

In order to access all of the privileges associated with the roles, Charles must set either Role2 or ALL as his current session (by using SET ROLE ROLE2 or SET ROLE ALL statements). Charles now has the privileges listed in the following table.

Because DBADMIN previously submitted...	Charles can now...
`GRANT SELECT, UPDATE, INSERT, DELETE ON Accounting.AccPay TO Role2 WITH GRANT OPTION;`	select, update, insert, or delete rows in the AccPay table.
`GRANT Role1, Role2 to Charles WITH ADMIN OPTION;`	grant, revoke, or drop Role1 or Role2 as needed.

Administrative Procedures for Roles

The following table lists the SQL statements that you can use to manage users using roles.

IF you want to ...	THEN use the...
control privileges at the group level	following procedure: **1** Log on as user DBC and grant to your database administration user (such as DBADMIN) the CREATE ROLE privilege, including the WITH GRANT OPTION. For example: `GRANT CREATE ROLE TO DBADMIN WITH GRANT OPTION;` **2** Log off as user DBC and log on again as your database administrator. **3** Use CREATE ROLE *rolename* statements to define one role for each set of privileges per group. **4** Grant the appropriate privileges to each role. **5** Grant one or more roles to one or more users until all users in all groups have the necessary privileges.
assign a role as the standard session default for a user	DEFAULT ROLE=*rolename*/NONE/NULL/ALL option of the CREATE USER or MODIFY USER statement. **Note:** The role must have already been explicitly granted to the receiver, except when the user submitting the CREATE or MODIFY statement has been granted that role, or has an active role (current role plus nested roles) which has been granted that role with the WITH ADMIN OPTION. (This also must be the case to assign a role at user creation time.) `MODIFY USER` username `AS DEFAULT ROLE=NULL;` A newly assigned default role does not affect the current role of an active session; it takes affect when the user next logs on.
let another user grant or drop roles	WITH ADMIN OPTION phrase when you grant the role: `GRANT rolenameA TO username WITH ADMIN OPTION;`
find out what role is current for this session	SELECT ROLE statement:
disable or change your current role for this session	SET ROLE *rolename*/NULL/NONE/ALL statement; for example, to disable the default role for the rest of the session, use SET ROLE NONE. If you specify a specific role, the role must exist and have already been granted to you. **Note:** The SET ROLE statement is treated as a DDL statement. It is permitted if it is the only statement or the last statement of a transaction. It is not supported within a stored procedure.
find out about role and user GRANT activity	DBC.AllRoleRights and DBC.UserRoleRights views. For column details, see *Data Dictionary.*
remove all privileges of a role from all members	DROP ROLE *rolename* statement.
remove a user from membership in a role	REVOKE *rolename* FROM *username* statement.

Using Profiles to Manage Settings

Profiles simplify the management of user settings such as accounts, default database, spool space, temporary space, and password attributes for a group of users.

Assign specific settings to a profile, then assign a profile to a user in the CREATE USER or MODIFY USER statement. This way, instead of managing settings for users on an individual basis, you can modify the settings for a profile to immediately apply the changes to all users that use the profile. You can also assign a user to a different profile any time using the MODIFY USER statement.

All users (except DBC) must be explicitly granted the CREATE PROFILE privileges in order to create, drop, grant, and implement a profile. For more information, see "Granting Create Privileges to a New User" on page 129.

When using profiles with directories, the directory administrator assigns external profiles to directory users. External profiles are essentially the same as database-assigned ones. The only difference lies in how they are assigned to users. Instead of using a CREATE USER or MODIFY USER statement to assign a profile, the profile is assigned by the administrator when setting up the schema of the user. For more information on integrating directories with your Teradata Database system, see *Security Administration*.

Creating Profiles

When you create a profile, you automatically receive creator privileges (implicit privileges granted through ownership do not apply). This gives you the DROP PROFILE privilege on a profile.

Creator privileges do not enable you to assign a default profile to a user. These parameters are specified in the PROFILE clause of CREATE/MODIFY USER, for which you need the CREATE USER or DROP USER privilege, respectively.

If a directory-based user is mapped to a database-defined user, then the method for creating profiles is the same. The SQL to create both directory and database assigned profiles are the same. (This is not, however, the case for roles.)

To create a profile assigned to a database, follow this procedure:

1. Log on as user DBC or as the administrative user.
2. Grant the CREATE PROFILE privilege using WITH GRANT OPTION to your database administration user; for example:

   ```
   GRANT CREATE PROFILE TO DBADMIN WITH GRANT OPTION;
   ```

3. Log off as user DBC. Then log on again as your database administration user (DBADMIN).
4. Create a profile that defines the following:
   - A profile name
   - One or more of the following:
     - PASSWORD attributes

- SPOOL space
- TEMPORARY space
- ACCOUNT (one or more IDs, with or without Priority Group and ASE codes)
- DEFAULT DATABASE

5 To assign the profile to a user, use the AS PROFILE modified in a CREATE USER or MODIFY USER statement:

```
MODIFY USER username AS PROFILE=profilename ;
```

To remove a profile from a member but retain the profile itself:

```
MODIFY USER username AS PROFILE=NULL ;
```

Implementing Profiles

The assignment of a profile to a group of users ensures that all members of that group operate with a common set of parameters. The values in a profile always take precedence over values defined for a user through the CREATE and MODIFY USER statements.

For example, you may create a profile by applications such as tactical queries, strategic queries, database maintenance, or database loading. Or, create profiles by user area such as web user, IT, ad-hoc user, power user, application developer, or by business divisions.

With profiles, you can manage the following attributes:

- Password settings, including:
  - Days before expiration
  - Number of minimum and maximum characters
  - Number of allowable logon attempts
  - Duration of user lockout (indefinite or elapsed time)
  - Number of times allowed for password reuse
  - Requiring mixed case, use of alpha, numeric, and/or special characters
  - Allowing or disallowing username within the password string
- Account strings, including ASE codes and Performance Groups
- Default database assignments
- Spool and temporary space limits

In a profile, the SPOOL and TEMPORARY limits may not exceed the current space limits of the user submitting the CREATE/MODIFY PROFILE statement.

All members inherit changed profile parameters. The impact is immediate, or in response to a SET SESSION statement, or upon next logon, depending on the parameter:

- SPOOL and TEMPORARY space allocations are imposed immediately. This affects the current session of any member who is logged on at the time his or her user definition is modified.
- Password attributes take effect upon next logon. For attribute details, see "Customizing Your Password Controls" on page 186.

- Account IDs and a default database are considered at next logon unless the member submits a SET SESSION DATABASE or SET SESSION ACCOUNT statement, in which case the specified databasename or account ID must agree with a profile definition. For further information on accounts, see "Defining Accounts" on page 143.

When using profiles with directory users, the assignment to a profile must be done through the directory server schema of the user. Unlike roles, profiles are not classified as internal or external.

For more detailed information on integrating directories with your Teradata Database system, see *Security Administration*. For syntax on setting up profiles, see "CREATE PROFILE" in *SQL Reference: Data Definition Statements*.

Profile User Defaults

Profile definitions apply to every assigned user, overriding specifications at the system or user level. However, any profile definition can be NULL or NONE.

If you do not specify any profile definitions, the default value for the user session is determined as listed in the following table.

<table>
<tr><th>IF the following is null ...</th><th>THEN the default value is taken from ...</th></tr>
<tr><td>a password attribute</td><td>the system-wide specification for the corresponding attribute in the DBC.SysSecDefaults table, as explained in "Customizing Your Password Controls" on page 186.</td></tr>
<tr><td>all other parameters</td><td>definitions in the CREATE USER or last MODIFY USER statement.
If none are specified, a value is determined as follows:
<table>
<tr><th>Parameter</th><th>Default Value</th></tr>
<tr><td>Account ID</td><td>The default account ID of the user.
If you did not specify an account when you created that user, the account of its owner is used.</td></tr>
<tr><td>Performance group</td><td>Level M.</td></tr>
<tr><td>DEFAULT DATABASE</td><td>The setting defined for the user.</td></tr>
<tr><td>SPOOL</td><td>The same SPOOL value as the immediate owner of the space in which the user is being created.</td></tr>
<tr><td>TEMPORARY</td><td>The setting defined for the user.</td></tr>
</table></td></tr>
</table>

Dropping Default Database, Role, or Profile

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 database, role, or profile, the default for each affected user is handled as described in the following table.

IF the dropped object is a ...	THEN Teradata Database ...
default database	uses the username space by default. The user can use the SET SESSION DATABASE statement to set the default during a session.
default role	no longer uses that role by default for privilege checking when the user logs on. Use the SET ROLE ALL statement to reactivate all remaining roles within a session. **Note:** Use REVOKE to remove one or a few users from the membership of a role. Use DROP ROLE to remove all members from a role.
profile	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.SecurityDefaults (see "Customizing Your Password Controls" on page 186) **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.

Defining Accounts

A session is always associated with an account. At logon, the session is associated with the default account of the user unless the logon or startup string specifies a different account. During a session, you can submit a SET SESSION ACCOUNT statement to change to a different account, provided that the specified account is already defined in the most current CREATE/MODIFY USER/PROFILE statement. For examples on how to set session accounts dynamically, see "SET SESSION ACCOUNT" in *SQL Reference: Data Definition Statements*. For more information on what counts as the default account, see "Finding the Default Account" on page 145.

Accounts are useful for the following:

- Managing workloads, especially when used with priority scheduling or Teradata DWM.

- Monitoring resource usage for performance tuning and anticipating capacity requirements.
- Billing purposes, particularly when a user account is associated with one or more Account String Expansion (ASE) codes, which can report session activity with a fine granularity. For a database, you can specify an account to charge for the allocated space. For a user or profile, you can specify one or more accounts to charge for the space a user occupies, or the resources used during sessions the user initiates.

You can assign or modify a user account string with a performance group name ($*groupname*$), or an ASE designation (&*char*), or an account ID. Enclose the entire string in single quotes:

```
'$M$acct101&D'
```

If you assign more than one account, separate the strings with a comma and enclose the complete definition in parentheses as shown in the following example:

```
('acct407&D&T','acct407')
```

You can define account strings at the user level or the profile level. A profile enables you to assign or change all or any part of the designation just once for many users; for example:

```
CREATE PROFILE tactqry AS ACCOUNT = '$H2$cacct101&H';
MODIFY USER user1,user2,user3 AS PROFILE = tactqry;
MODIFY PROFILE tactqry AS ACCOUNT = '$M2$acct101&H' ;
```

If you define accounts at the user or profile level, the user can specify the account string to be used either at the time of logon or during the session.

Naming your Account ID

Enhance the results of query and workload analysis by using Account IDs in reports from other utilities and features. You can join the account id with the account id field in the DBQL tables to get more detailed information about the requests being accounted for in the DBC. AMPUsage table.

For maximum accounting effectiveness and collection at the granularity of individual session or request level, use the naming standards in the following table for your account ID.

The following request type...	Should have the naming standard...	And will be expanded as...
sub-second	$XX$YYY+&S	(17 bytes) $XX$YYY+sssssssss
batch utilities	$XX$YYY+&S	(17 bytes) $XX$YYY+sssssssss
other requests and stored procedures	$XX$YYY-&l	(30 bytes) $XX$YYY+llllssssssssssrrrrrrrr

where:

- $XX$ is the performance group. (For more information on performance groups, see "Priority Scheduler" in *Utilities*.
- YYY is the account name.
- llll is the host number.
- sssssssss is the session number.
- rrrrrrrr is the request number.

The ampersand '&' designates an account string expansion (ASE) variable. For more information, see "Tracking Accounts With Account String Expansion (ASE)" on page 148.

Join the account ID with the account string field in DBQL tables to get further query reporting. Join sub-second queries with the DBQLSummaryTbl for additional analysis. Join batch utility and stored procedure requests with DBQLogTbl and DBQLSQLTbl.

Finding the Default Account

Each time a CREATE/MODIFY DATABASE statement is processed, a row is inserted or updated in the system table DBC.Accounts and in DBC.DBase. Each time a CREATE/MODIFY USER/PROFILE statement is processed, a row is inserted or updated in DBC.Accounts and DBC.DBase or DBC.Profiles. The initial default account is determined as described in the following table.

<table>
<tr><th>IF ...</th><th>THEN the default account is ...</th></tr>
<tr><td>the user has a profile with one or more accounts</td><td>the first account defined in the profile.
<b>Note:</b> A profile account takes precedence over the user account.</td></tr>
<tr><td>the user has a profile with no accounts</td><td>the first account defined in the user or database definition, if any.</td></tr>
<tr><td>the user has no profile</td><td>the first account defined in the user or database definition, if any.</td></tr>
<tr><td>no account is defined for the user</td><td>determined as follows:
<table>
<tr><th>IF no account is specified for...</th><th>THEN the default is ...</th></tr>
<tr><td>a database</td><td>the account of the immediate owner of the database.</td></tr>
<tr><td>a user without a profile assignment</td><td>the account of the immediate owner of the user.</td></tr>
<tr><td>a profile</td><td>none for the profile itself.
A NULL is inserted in DBC.Profiles.DefaultAccount.</td></tr>
<tr><td>members of a profile with a NULL account</td><td>the first account in the user definition, if any; otherwise, the account of the immediate owner of the user.</td></tr>
</table>
</td></tr>
</table>

IF ...	THEN the default account is ...
multiple accounts are defined	the first account in the definition string. **Note:** The remaining definitions are stored in the DBC.Accounts table to be used for validation of user-specified accounts.

For example, if you assign a user with an account string but the user is assigned to a profile, then the account defined on the profile will always be used. If, however, you remove the profile from the user or remove the account string from the profile, the account defined on the user itself is used. If you did not specify an account when you created that user, the account of its owner is used.

The rest of this section discusses defining an account priority, changing the priority of a query, finding session statistics, and defining ASE variables. For syntax details, see the ACCOUNT keyword under "CREATE USER" and "CREATE PROFILE" in *SQL Reference: Data Definition Statements.*

Accounts and Performance Group Names

A session always runs under an account ID and a Priority Scheduler performance group. The performance group portion of the account string indicates the relative service priority for the account. Performance group names, their relative levels of service, and other related parameters are either:

- Assigned by default
- Defined by you as customized variables, per user or profile (based on the parameters you established using the Priority Scheduler schmon or the Priority Scheduler Administrator of Teradata Manager; for details, see *Utilities* and *Teradata Manager User Guide*).

  In the following example, note the definitions for ACCOUNT:

`CREATE USER DBA01`	Name of the user being created (DBA01)
`,FROM DBA_PROF`	Present owner of space
`,AS PERManent=0`	Amount of permanent space
`,SPOOL=1000000`	Amount of spool space
`,PASSWORD=DBA01`	Password logon protection
`,FALLBACK`	Table data protections default
`,ACCOUNT='$M2$acct101&H'`	*`$M2$`* - A specified Performance Group name *`acct101`* - account ID *`&H`* - Hour ASE variable, unqualified with &D. Statistics collected for a specified hour on one day will be summed with existing statistics for the same hour on other days.

When you log on to Teradata Database, the performance group for the initial session is as described in the following table.

IF a …	THEN the session runs under …
custom account is specified or determined by default that includes a performance group	that performance group. (For more details, see "Finding the Default Account" on page 145.)
performance group is not defined for the account under which the session is initiated	performance group M.

Once a session is active, you may change the associated account during runtime. You may also change the performance groups regardless of if it was initially set by specification or by default at any time. You can do this at either the session level or the query level.

Thus, at any point in time, every session and its executing query run under:

- The performance group established at logon time
- The performance group that the user modified dynamically by:
  - Issuing a Teradata SQL SET SESSION ACCOUNT...FOR [SESSION/REQUEST] statement, or a Performance Monitor SET SESSION ACCOUNT request, during the current session.
  - Embedding Teradata SQL SET SESSION ACCOUNT statements in the executing program or BTEQ script.
- The performance group you dynamically impose on the current:
  - Session, via Teradata Manager Performance Monitor session control.
  - Request or session, using the Teradata Performance Monitor program or the PM/API SET SESSION ACCOUNT request.

Viewing Account Strings and Session Statistics

The DBC.AMPUsage view provides cumulative information about the use of each AMP for each user/database and account during every session. Each new name results in a new set of statistics. If the account is dynamically changed during processing, a separate row is generated for each account.

The underlying system table is DBC.Acctg. Updates to the table are made periodically on each affected processor, continually adding to what is already in the table until you reset the counters (see "Clearing DBC.Acctg Table with DBC.AMPUsage View" on page 121).

DBC.AMPUsage provides aggregated information by AMP, username, or accountID. You can include Account String Expansion (ASE) variables in the account string to provide more granularity. This can increase the usefulness of DBC.AMPUsage data for capacity planning and performance tuning, and in charge-back and accounting software.

A few examples of possible charge types are described in the following table. (For details, see "AMPUsage" in the chapter titled "System Views: Usage and Examples" in *Data Dictionary*).

Charge	Comment
Space occupied by the default databases this user employed during the session.	If a default database is not defined for a user, the default is used as explained in "CREATE USER Default Values" on page 128.
Time consumed by the session under this account.	If no account string is specified at logon, the default is used as explained in "Finding the Default Account" on page 145. Also, the session may use several accounts during runtime.
Processing resources consumed by the session or sessions initiated under this account.	You can design billing algorithms by individual or user profile account. Include ASE codes for performance tuning and capacity planning, so you can measure the amount of resources used for different workloads for different days and times. For details, see "Tracking Accounts With Account String Expansion (ASE)" on page 148.

Tracking Accounts With Account String Expansion (ASE)

Account String Expansion (ASE) allows you to more precisely measure individual SQL queries and get increased granularity of AMP usage measurements (ASE does not modify the AMP usage statistics gathering process). You can gather statistics for capacity planning, gather information for use in charge back and accounting, track resource consumption, or trace bad queries to their source.

ASE is optional. You must explicitly modify logon account strings with ASE variables to activate ASE accounting. If activated, Teradata Database will collect statistics based on those variables, and store the information into the DBC.AmpUsage during the active session.

At the finest granularity, ASE can be used to generate a summary row for each query per AMP. You can direct ASE to generate, per AMP, a row for each user, each session, or per aggregation of the daily activity for a user. You can also generate individual rows and use Teradata Manager to provide the summary functions, for example, one summary row per day per hour, by user, or by account.

Enabling ASE

Before a user can activate any of the ASE functions, there must be a matching account string in the DBC.Users 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 user account string or to the profiles which 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 =
'finance&D&L', '$H$finance&L';
```

The first account listed is the default account string. Depending on how the account is listed, ASE usage is handled as described in the following table.

If the DBA submits the following...	John can...
`MODIFY USER john AS ACCOUNT = ('finance&D&H', 'finance');`	automatically log in with date and hour ASE variables activated since the variables are defined in his default account string. He can log on and not use the ASE variables with: *.logon john, mypassword, 'finance';*

`MODIFY USER john AS ACCOUNT = ('acct101', 'finance&D&L');`	manually activate ASE substituting for date and logon time by submitting:
	.logon john, mypassword, 'finance&D&L';
	SET SESSION ACCOUNT='acct101&D&L';

`MODIFY USER john AS ACCOUNT = ('acct123', 'acct246');`	never use ASE at any time. He cannot dynamically try to logon with ASE variables in the account string until the DBA modifies his logon string to allow ASE.

Some of the more commonly used ASE variables include date (&D), time (&T), hour (&H), time of log on (&L), or a combination thereof. For more information on the different variables, see "Coding ASE Variables" on page 151.

Setting the Default Account String to Use ASE

ASE is an optional feature. To enable it, define one or more of the ASE variables in an account string either directly in the user definition, or in the definition of the profile assigned to the user. You can also 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 ACCOUNT=('acct407&D&H','acct407');
```

If TAW logs on using the default account string, date and hour monitoring occur for each SQL request. At query execution time, Teradata Database replaces &D with the current date, and replaces &H with the current hour.

To turn off monitoring, TAW must do one of the following:

- Enter *'acct407'* each time he logs on
- Change the account during a session with:

  ```
  SET SESSION ACCOUNT='acct407';
  ```

In another case, you assign two account strings to user DTG. However, the first (default) account string does not specify date and hour expansion:

```
MODIFY USER DTG ACCOUNT = ('acct101', 'acct101&D&H');
```

If DTG logs in using the default account string, date and hour logging do not occur. To initiate them, DTG must type *&D* and *&H* in an account string, either when he logs in or during the session:

- Log in using *.logon DTG, mypassword, 'acct101&D&H'*
- Change the account during a session with: *SET SESSION ACCOUNT='acct101&D&H'*

In another case, assume you omit the ASE variables in the two account strings for user AM1:

```
MODIFY USER AM1 ACCOUNT=('acct101', 'acct102');
```

Therefore, AM1 cannot invoke the ASE feature at any time.

Resource and Performance Considerations

ASE has a negligible effect on parsing engine (PE) performance. The cost incurred for analyzing the account string requires only a few microseconds. However, AMPs have the burden of additional logging in DBC.AMPUsage. Depending on the number of users and the ASE options chosen, the added burden may vary from slight to enough to degrade overall performance.

The information collected when using ASE can be very helpful in analysis, but you should take care not to create a bigger problem than you are trying to solve. For this reason, Teradata recommends that you monitor ASE usage carefully. Clean up DBC.AmpUsage regularly as it can fill up quickly.

When deciding whether to use ASE, consider the following:

- You must determine the measurement rate you need and the users you wish to monitor.
- Each different user/account string or profile/account string pair results in a new row being inserted in AMPUsage.
- Collection activity occurs on all AMPs involved with the request.
- Performance impact of ASE can vary greatly depending upon granularity requested and the types of requests submitted.
- Use only the ASE variables required to capture the information you need.
- Be sure to summarize and save history rows as needed and then clear the DBC.Acctg accumulators (see "Clearing DBC.Acctg Table with DBC.AMPUsage View" on page 121).

Use &T With Caution

When &T is specified, ASE logs a row to DBC.AMPUsage for every AMP, for every request. The I/O activity varies dramatically from site to site based on the number of concurrent users, their account strings, and the usage of the ASE options.

Using &T can be very resource intensive so Teradata strongly recommends that you do not use &T:

- With OLTP applications
- With TPump
- In the default account string

If you notice an impact on system performance while using &T, delete rows from DBC.Acctg and discontinue using &T.

For more discussion on the performance impact of ASE, see *Performance Management*.

Coding ASE Variables

When you enable ASE accounting, the system expands the variables and substitutes values for the variables at logon or query execution time. For example, $M$12345&D becomes $M$12345040704 for a logon at July 4, 2004 under the 12345 account. (The presence of $R$, $H$, $M$, or $L$ priority codes are ignored by ASE.)

The ASE variables may be used in any combination and in any order, subject to the constraints on length and position. These are:

- The maximum length of an account string is 30 characters even after expansion. If the expansion of the account string goes beyond 30 characters, the system truncates the excess characters from the right hand side of the string.
- If priority codes are being used, the first ASE variable must begin after the priority codes. If no priority codes are used, the first account string expansion variable can begin in any character position of the account string.

The ASE substitution variables are as described in the following table.

Variable	Format	Description
&D	YYMMDD	Date substitution. The system substitutes the date it received the SQL request into the account string. **Note:** If &D is in position 26 or higher of the account definition, truncation occurs. You can take advantage of this truncation to monitor resources on a yearly or monthly basis.
&H	HH (24 hour clock)	Hour substitution. 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 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. **Note:** If &H is in position 30 of the account definition, truncation occurs.

Variable	Format	Description
&I	LLLLSSSSSSSSSRRRRRRRRR	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.
&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. In this case, use the &L option to generate separate statistics and to monitor the LogonSource field of DBC.LogOnOff. **Note:** If &L is in position 17 or higher of the account definition, truncation occurs.
&S	SSSSSSSSS	The system substitutes the current session number.
&T	HHMMSS (24 hour clock)	Time substitution. The system substitutes the time of day it received the SQL request. **Caution:** 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. 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. If you use the &T variable without the &D variable, the system sums statistics collected for a specified time on one day with existing statistics for the same time on other days. **Note:** If &T is in position 26 or higher, truncation occurs. You can use this truncation to monitor resources hourly or by the minute.

Restrictions and Usage Rules for ASE Variables

The following restrictions and cautions apply to ASE variables:

- Account strings cannot exceed 30 characters. If the expanded account string exceeds 30 characters, characters to the right of position 30 are truncated.
- Be aware that the account string 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 and is truncated to the right of position 30.
  - Performance group names (although ASE ignores them).
- You can intersperse ASE variables with literals, subject to the constraints of length and of position relative to performance group controls (see "Accounts and Performance Group Names" on page 146).
- When combining ASE variables, keep the following in mind:
  - You can use multiple ASE variables in any order in the account string following the first ASE variable.
  - Some combinations generate strings long enough to truncate true account strings.
  - Although it is perfectly legal to combine the &H and &T options, the information collected is redundant.
  - 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 Database interface to log on, including:

- BTEQ
- FastLoad
- MultiLoad
- TPump (except for &T)
- FastExport
- Teradata SQL Assistant (formerly known as Queryman)

The exceptions are as described in the following table.

Do not use ...	With ...	Because ...
any ASE code	PMPC statement	requests generated are not parsed and substituting variables cannot be expanded.
	ARC	ARC generates two kinds of requests: • SQL type requests that go through the parser and expand the ASE variables. • Direct requests from ARCMAIN to the AMP that bypass the parser. As a result, substitution variables are not expanded, so the rows in DBC.AMPUsage contain literal ASE codes rather than the expected date, hour, and timestamp values.
&T	TPump	&T generates a row in AMPUsage for nearly every SQL statement.

Suggested Clean Up When Using ASE

By adding even just one variable, the amount of information collected in to DBC.Acctg can greatly increase. Reclaim disk space by regularly updating DBC.Acctg to clear out values and reset the accumulators. You can use Teradata Manager to first summarize and save historical data. (For more information and examples, see "Clearing DBC.Acctg Table with DBC.AMPUsage View" on page 121.)

System Accounting Views

Teradata Database provides the following views to support administration of accounts and accounting functions:

- DBC.AccountInfo
- 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.

You can also use the accounting features of Teradata Manager to collect data and report on system usage. Teradata Manager automatically manages system accounting tables historically and provides charts and graphical views of the accounting data. For more information on data collection and trends reporting, see *Teradata Manager User Guide* and the online help for Teradata Manager.

DBC.AccountInfo

The DBC.AccountInfo 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.AccountInfo to find out:

- What accounts are valid for which user
- The assignment of:
  - Priority Scheduling codes
  - ASE codes

This dictionary table ...	Stores ...
DBC.Accounts	all accounts for all databases, users, and profiles. If an account is not specified at creation time, the default is used. The default is determined as explained in "Finding the Default Account" on page 145. 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 • 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. **Note:** If an account is not specified for a profile, the value is NULL in the DefaultAccounts field for that profile.

Example

The following query selects all account strings with a RUSH priority code, and whether the name associated with an account is an individual user or a profile:

```
SELECT AccountName,Name,UserOrProfile
FROM DBC.AccountInfo
WHERE AccountName LIKE '$R%'
ORDER BY AccountName ;
```

In this example, the view returns:

```
AccountName        Name                 UserOrProfile
--------------     ---------------      -------------

$R_AR1022          AcctsRecv            Profile
$R_P2450           DBADMIN              User
$R_P1230           DBC                  User
$R_P1230           SysAdmin             User
$R_P3450           SystemFe             User
```

DBC.AMPUsage

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 time 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, accountID, 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 “Clearing DBC.Acctg Table with DBC.AMPUsage View” on page 121).

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 if one or more tables has skewed row distribution across AMPs.
- Determine which session caused reduced performance.
- 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 Chapter 13: "Tracking Processing Behavior with the Database Query Log (DBQL)."

For a look at up-to-the-moment activity in near-real-time, you can use the Teradata Performance Monitor, as discussed in the Teradata Performance Monitor online help or in *Teradata Manager User Guide*.

Example: Totalling 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	$M$P9210	6,336.76	505,636
DBA01	$H$P9210	4,387.14	303,733
DBA01	$R$P9210	1.28	166

For detailed information on these and all the system views, see *Data Dictionary*. For more information on how to use DBC.AMPUsage and other views to find problems and improve performance, see *Performance Management*.

System Accounting Functions

System accounting serves three important administrative functions:

- Charge-back billing (for equitable cost allocation)
- Capacity planning (to anticipate your needs)
- Resource management (to identify performance anomalies)

For more information on how to identify and analyze session behavior and apply resource control, see Chapter 12: "Tools for Managing Resources." For details on optimizing the system through workload and session management, see *Performance Management*.

Charge-back Billing

You may need to charge users for their use of resources. Charge-back billing permits equitable cost allocation of system resources across all users.

The user account string enables you to summarize resource usage by accountID. The system table DBC.Acctg tracks CPU 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.

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.

Resource Management

System accounting, as part of system management and in conjunction with the Index Wizard and other query analysis tools, can help you identify potential problem areas, such as unbalanced row distribution or inefficient join indexes.

Also, you may need to control who gets specific resources. Use the Priority Scheduler to manage user account priorities to maintain efficient operation of the Teradata Database while providing equitable service to multiple client utilities and users.

CHAPTER 6 Controlling and Tracking Privileges

This chapter provides information on controlling and tracking access to your Teradata Database. Topics discussed include the different types of privileges and how to grant and revoke them. It also discusses how to grant privileges to multiple users through cascading privileges or roles.

Overview of Access Privileges

Privileges[1] control, or authorize, the types of activities you can perform during a session. Privileges are maintained in the Data Dictionary through the DBC.AccessRights table which contains information about privileges granted explicitly or automatically to users, databases, PUBLIC, and roles as rows. Rows are inserted into or removed every time there is a:

- CREATE and DROP statement
- GRANT and REVOKE statement

Simplify management of privileges and reduce I/O to DBC.AccessRights by using roles. A role is simply a collection of explicit privileges. You can grant explicit privileges to a role and then grant the privilege to use the role to a user or a group of users.

Caution: While the GIVE command affects ownership and space, it is not reflected in the DBC.AccessRights table. GIVE should be used only with caution.

A user or database may have both implicit and explicit privileges. Additionally, a user may inherit explicit privileges through a role or PUBLIC. A user can also inherit explicit privileges from a role and its nested role, if any exist.

Teradata recommends that you grant users only the privileges they require for their job. Do not grant users more privileges than they need because this causes unnecessary checks and puts users at risk for unintentionally modifying or deleting database objects they should otherwise not access.

Warning on Altering Privileges for User DBC

Never alter access rights for user DBC. Doing so may cause installation, upgrade, maintenance, or archive procedures to end abnormally and consequently require Teradata Support Center to correct the problem.

[1] Privileges are sometimes referred to as rights, access rights, or permissions. This book uses these terms interchangeably.

Explicit Privileges

An explicit privilege is the right to take action on (access) an object or another database or user, as granted by one user (the grantor must have the right to issue the GRANT option) to another; for example:

```
GRANT SELECT ON UserA.TestTable TO UserB;
```

Or, the system automatically grants explicit privileges to a user or database when it creates a new object (some privileges are given to the creator and some to the created user or database). For example, a newly created user is automatically given the right to create a table in his or her own space, and the creator of a table is automatically given the right to alter or drop that table.

Both explicitly granted and automatically granted privileges are logged in the DBC.AccessRights table and can be revoked. Note that DIP scripts include GRANT statements that grant privileges.

Note: When granting privileges, the grantor does not need to have any privileges (or WITH ADMIN OPTION) on the grantee role, user, or database in order to grant it a privilege. For example, a user without any privilege on a role can grant privileges to that role. However, the grantor does need privileges (with grant option) on the *objects* that are being granted.

Entries in DBC.AccessRights can be retrieved through the DBC.AllRights view. This view returns all users who have explicit privileges, and the objects on which the privileges were granted. For details, see "Dictionary Views and Tables for the Administrator" on page 114.

The right to confer the GRANT option to a third or subsequent user can be conferred using the WITH GRANT OPTION of the GRANT statement.

For example, if UserA creates TestTable in his own space and then grants UserB the right to select data from that table, UserA can also grant to UserB the right to grant the SELECT privilege to other users.

If UserA submits the following statement:

```
GRANT SELECT ON UserA.TestTable TO UserB WITH GRANT OPTION ;
```

UserB has the right to grant SELECT on UserA.TestTable to UserC or to UserD, and so on.

Views That Report Explicit Privileges

The system stores explicit privileges as rows in the DBC.AccessRights or DBC.RoleGrants tables when a user submits a GRANT or CREATE statement. The following table lists several views that offer information about the explicit privileges granted automatically or explicitly to a user, database, role, or PUBLIC.

For information about ...	Use the view named ...
all explicit privileges that have been granted	DBC.AllRights
explicit privileges the requesting user has granted to other users	DBC.UserGrantedRights

For information about ...	Use the view named ...
explicit privileges the requesting user has been granted	DBC.UserRights
all roles directly granted to the requesting user	DBC.RoleMembersX
each role and every user or role to whom it has been granted	DBC.RoleMembers
all explicit privileges granted to each role	DBC.AllRoleRights
all explicit privileges granted to each role for the requesting user	DBC.UserRoleRightsX

For More Information

For more information about explicit privileges, see the following references.

IF you want more information about...	THEN see...
the GRANT statement, privilege types, using roles, and how to control user privileges	• "Granting Privileges" on page 172 • "Granting a Set of Privileges to Multiple Users" on page 180 • *Security Administration*
how to set up and maintain a secure database environment	*Security Administration*
views that return information about privileges	*Data Dictionary*
authorization for CREATE PROCEDURE	*SQL Reference: Data Definition Statements*
authorization for CREATE FUNCTION	*SQL Reference: UDF, UDM, and External Stored Procedure Programming*
authorization for and how to use the performance monitoring tools	• *Workload Management API: PM/API and Open API* • *Performance Management* • *Teradata Manager User Guide*
authorization for and how to set up and use the resource usage monitoring tools	*Resource Usage Macros and Tables*

Inherited Privileges

A user may have an explicit privilege either because the privilege is specific to the user or by inheritance through a role or PUBLIC that has the explicit privilege.

For more information on granting privileges to roles, see "Rules for Using Roles" on page 136. For more information on granting privileges to PUBLIC, see "SQL Data Control Language Statement Syntax" chapter of *SQL Reference: Data Definition Statements*.

Implicit Privileges

Implicit privileges are privileges users have on objects because they own those objects, either directly (the immediate owner) or indirectly. Implicit privileges are sometimes called ownership privileges.

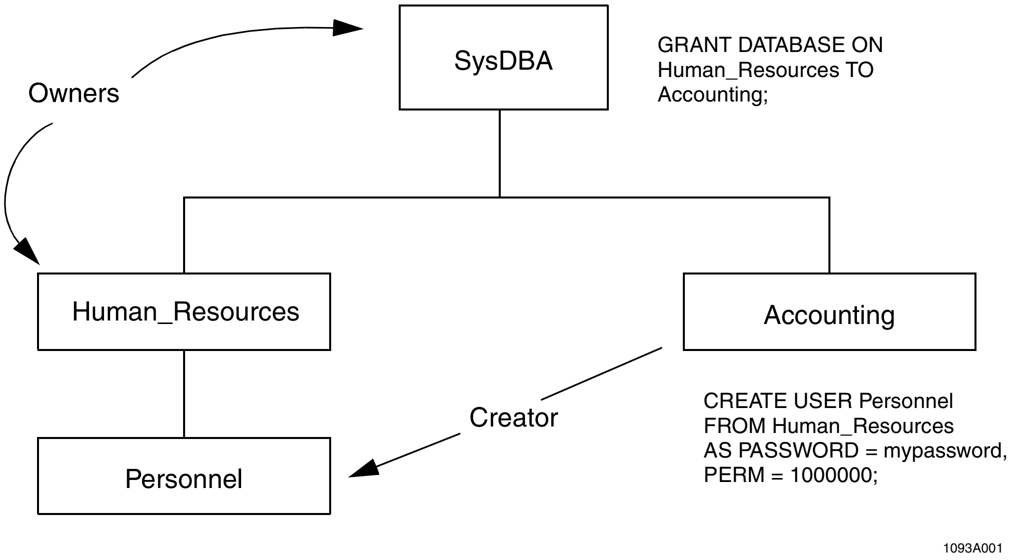
A database or user has implicit privileges on an object as long as the database or user is an owner of the object. However, you can use the GIVE statement to change the ownership hierarchy (see "Changing the Hierarchy with GIVE" on page 33). Implicit privileges on an object cease once the object is dropped.

Implicit privileges cannot be revoked and they are not logged in DBC.AccessRights. If John owns SampleTable and revokes all access privileges from all users, including himself, no one can access SampleTable. Not even John can access the table even though he owns it. However, because owning SampleTable means he has implicit privileges on it, John can grant explicit privileges back to himself at any time.

Note that there are a few implicit privileges that do allow access to an object without having an explicit privilege.

For more information on privileges and the GRANT statement, see Chapter 6: "Controlling and Tracking Privileges" or "SQL Data Control Language Statement Syntax" in *SQL Reference: Data Definition Statements*.

Example of Privileges



In the diagram above, Accounting is the creator. The system automatically inserts rows for explicit privileges in the DBC.AcessRights table for the creator (Accounting) and for the created user (Personnel). These privileges can be revoked.

Personnel is the created object. Personnel automatically receives some but not all explicit privileges on itself.[2] The privileges Personnel does receive are inserted automatically in DBC.AccessRights. These privileges can also be revoked.

Human_Resources is the immediate owner. The database does not insert any rows into the DBC.AccessRights table for Human_Resources and SysDBA. However, Human_Resources and SYSDBA have implicit privilege, as owners, to grant themselves explicit privilege over Personnel. You cannot revoke the implicit privilege to GRANT explicit privileges over owned objects.

Privileges Automatically Received

As a creator, you automatically receive specific privileges. However, a created user or database may need to be explicitly granted privileges not automatically granted upon creation. When you create a database or user, you may also have to additionally grant privileges listed below.

Automatic Privileges For Creators

The creator of a database or user is automatically granted the following privileges on the database or user he or she just created. They include the following 24 privileges:

- 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

[2] The explicit privileges Personnel does not receive includes privileges listed under "Automatic Privileges For Created User/Database" on page 164.

- EXECUTE
- INSERT
- SELECT
- RESTORE
- UPDATE

Automatic Privileges For Created User/Database

The created user or database gets the following 20 privileges automatically on itself.

- 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
- RESTORE
- UPDATE

Note: The privileges a new user/database receives once created is the same as those listed for automatic creator above ***except***:

- CREATE USER
- CREATE DATABASE
- DROP USER
- DROP DATABASE

The created user/database *does not* get the CREATE USER/DATABASE or DROP USER/ DATABASE on itself.

Privileges That Must Be Explicitly Granted

The following section lists the privileges a creator must grant to itself or that a user or database must be granted by someone else.

For Creators

The creator does not get the following eight privileges automatically on the created user/ database. They must explicitly be granted:

- ALTER EXTERNAL PROCEDURE
- ALTER FUNCTION
- ALTER PROCEDURE
- CREATE EXTERNAL PROCEDURE
- CREATE FUNCTION
- CREATE PROCEDURE
- EXECUTE FUNCTION
- EXECUTE PROCEDURE

For Created User/Database

The created user or database does *not* get the following 12 privileges automatically on itself; they must be granted explicitly:

- ALTER EXTERNAL PROCEDURE
- ALTER FUNCTION
- ALTER PROCEDURE
- CREATE DATABASE
- CREATE EXTERNAL PROCEDURE
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE USER
- DROP DATABASE
- DROP USER
- EXECUTE FUNCTION
- EXECUTE PROCEDURE

These are the same as the eight privileges a creator must be explicitly granted plus the four excluded privileges (CREATE USER/DATABASE and DROP USER/DATABASE).

Additional Privileges

The following privileges must be granted by DBC or another user who already has these privileges:

Table level privileges

- INDEX
- REFERENCES

Monitor privileges

- ABORTSESSION
- MONRESOURCE
- MONSESSION
- SETRESRATE
- SETSESSRATE

System level privileges

- CREATE PROFILE
- CREATE ROLE
- DROP PROFILE
- DROP ROLE
- REPLCONTROL

UDT privileges

- UDTMETHOD
- UDTUSAGE
- UDTTYPE

Description of Privileges

The AccessRight column of the following views:

- DBC.AllRights
- DBC.UserGrantedRights
- DBC.UserRights
- DBC.AllRoleRights
- DBC.UserRoleRights

reports a code that represents a privilege granted on a particular object. The following codes include:

```
AE = ALTER EXTERNAL PROCEDURE    D  = DELETE                MR = MONITOR RESOURCE
AF = ALTER FUNCTION              DA = DROP AUTHORIZATION    MS = MONITOR SESSION
AP = ALTER PROCEDURE             DD = DROP DATABASE         PC = CREATE PROCEDURE
AS = ABORT SESSION               DF = DROP FUNCTION         PD = DROP PROCEDURE
CA = CREATE AUTHORIZATION        DG = DROP TRIGGER          PE = EXECUTE PROCEDURE
CD = CREATE DATABASE             DM = DROP MACRO            R  = RETRIEVE/SELECT
CE = CREATE EXTERNAL PROCEDURE   DO = DROP PROFILE          RF = REFERENCE
CF = CREATE FUNCTION             DP = DUMP                  RO = REPLICATION OVERRIDE
CG = CREATE TRIGGER              DR = DROP ROLE             RS = RESTORE
CM = CREATE MACRO                DT = DROP TABLE            SS = SET SESSION RATE
CO = CREATE PROFILE              DU = DROP USER             SR = SET RESOURCE RATE
CP = CHECKPOINT                  DV = DROP VIEW             U  = UPDATE
CR = CREATE ROLE                 E  = EXECUTE (MACRO)       UU = UDT Usage
CT = CREATE TABLE                EF = EXECUTE FUNCTION      UT = UDT Type
CU = CREATE USER                 I  = INSERT                UM = UDT Method
CV = CREATE VIEW                 IX = INDEX
```

The following table lists the required privileges for some database objects or database setups.

To use the following...	You must have the following...
directory server integration	privileges from direct mappings to a permanent user or privileges granted to the system-defined default directory user EXTUSER. Integrated directories allow you to map directory users on directories such as Windows Active Directory and Sun Java System Directory Server to Teradata user and roles. Teradata remains the final authority for authenticating users and associating roles and profiles to users. The information in the directory is only a reflection of the Teradata Database system. Since no secure information, such as passwords, are kept in the Teradata extensions to the directory, anyone will be able to read information from the directory. However, only a validly authenticated directory administrator will be able to add, modify, or remove information in the directory. For more information on external authentication, encryption, confidentiality, and integrity services, see *Security Administration*.
external stored procedures	• CREATE EXTERNAL PROCEDURE privilege to create an external stored procedure. Creating an external stored procedure is not an automatic privilege for a user when creating a data or user. Also, it is not an implicit privilege already granted implicitly on the database and external stored procedure the database owns, unless the owner explicitly has this privilege on itself with WITH GRANT OPTION. • ALTER EXTERNAL PROCEDURE privilege to alter external stored procedures. The ALTER PROCEDURE statement with the C/CPP language clause can be used to change the execution mode or recompile existing external stored procedures (in case the current library is corrupted or the system has been reloaded). • This is not an automatic privilege for a user when creating a database or user. This privilege is retained by the DBA (initially only DBC has the implicit privilege) to be assigned only to production external stored procedures that have been completely debugged. **Caution:** The CREATE and ALTER EXTERNAL PROCEDURE privileges should be granted only to trusted developers. When an external stored procedure executes as part of the database application when running in unprotected mode on UNIX MP-RAS systems, it has the privileges of the root user. On Windows, it has the privileges of system mode. It can access or modify any file on the local disks even if the files are protected or read only. Be sure to debug and thoroughly check out any external stored procedures and verify that they do not compromise the system.
macros	CREATE MACRO, EXECUTE MACRO, REPLACE MACRO and DROP MACRO privileges. The immediate owner of the macro (the database in which the macro resides) must have the necessary privileges on objects named in the statements defined in the macro. CREATE MACRO and DROP MACRO privileges are required to create and drop macros. To use the REPLACE MACRO command for an existing macro, you must first have the DROP MACRO privilege. In addition, the user replacing a macro must have the privileges for all statements the macro performs.

To use the following...	You must have the following...
replication solutions	REPLCONTROL privilege. This privilege enables you to: • create, alter, and drop replication groups. • use the SET SESSION OVERRIDE REPLICATION statement to allow changes to replication group member tables in a state that would not normally allow changes to be made. REPLCONTROL, like the ROLE and PROFILE privileges, is a system privilege you can grant to a user, but not to specific tables or databases. User DBC is the only user who initially has the REPLCONTROL privilege but DBC can grant it to other users. For more information on the GRANT and REVOKE syntax for the REPLCONTROL privilege, see *SQL Reference: Data Definition Statements*. **Caution:** A user with the REPLCONTROL privilege can create a replication group consisting of any tables defined in the system. Grant REPLCONTROL only to trusted users.
stored procedures	CREATE PROCEDURE, DROP PROCEDURE, and ALTER PROCEDURE privileges. You must have the CREATE PROCEDURE privilege on the containing database or the user in which a stored procedure is to be created. The creator of a stored procedure is automatically granted the DROP PROCEDURE and EXECUTE privileges WITH GRANT OPTION on it. To recompile an existing internal stored procedure, you must have the ALTER PROCEDURE or DROP PROCEDURE privilege either on the named procedure or on the database containing the procedure. To grant a user the privilege to execute a stored procedure he or she did not create, submit: `GRANT EXECUTE PROCEDURE ON dbname TO username;`
triggers	SELECT, CREATE TRIGGER, and DROP TRIGGER privileges. To create a trigger, you must have the following privileges: • CREATE TRIGGER on the subject table or the database containing the table. • SELECT on any column referenced in the WHEN clause of the CREATE TRIGGER statement, or any column in a triggered action statement that requires read access for the execution of the statement. • The access rights normally required to execute the individual triggered SQL statements. To execute a trigger, you must have the following privileges: • The privileges required for executing the triggering statements. • The immediate owner of the trigger must have all the privileges of: • CREATE TRIGGER on the subject table or the database containing the table. • SELECT on any column referenced in the WHEN clause of the CREATE TRIGGER statement, or any column in a triggered action statement that requires read access for the execution of the statement.

To use the following...	You must have the following...
user-defined functions (UDFs)	privileges listed in the table below. Most of these privileges are not automatic and must be explicitly granted to users by the DBA.

To do the following...	You must have...
create a function	the CREATE FUNCTION privilege. **Caution:** UDFs execute as part of the database application. The database application is set up as a trusted user or system user. On MP-RAS, UDFs have all the privileges of the “root” user. On Windows, the database runs in system mode which means it can access and modify any file on the local disk even if the file is protected or read only. Therefore, do NOT give out this privilege to anyone not specifically developing UDFs.
display the C source of a function	at least one privilege on the function or the database containing the function. Use the SHOW FUNCTION statement to display CREATE/REPLACE FUNCTION text and the source code. If the user has the DROP FUNCTION privilege, the SHOW FUNCTION statement will display the C source. If the user does not have it, the C source is not displayed.
execute a function	the EXECUTE FUNCTION privilege on the database or specific function. To grant the user privileges to execute a specific user-defined functions, submit: `GRANT EXECUTE ON SPECIFIC FUNCTION` *`function_name`* `TO` *`username;`* To grant execute privileges on all UDFs in a database: `GRANT EXECUTE FUNCTION ON` *`dbname`* `TO` *`username;`*
comment on a function	the DROP privilege on the function.
get help on a function	at least one privilege on the function or the database containing the function.
• change the execution mode of a function • recompile or re-link a function	the ALTER FUNCTION privilege. A user-defined function running in protected mode runs in a process created to run as the user called "tdatuser." This user has no special operating system privileges. It is up to the site to decide what resources "tdatuser" is to have access to by setting access privileges to system resources as required by the site. **Note:** If you specify EXECUTE NOT PROTECTED, and the function fails during execution, the database could restart.
drop a function	the DROP FUNCTION privilege.
rename a function	the DROP FUNCTION privilege on the function of the containing database and the CREATE function privilege on the database containing the function.

<table>
<tr><th>To use the following...</th><th>You must have the following...</th></tr>
<tr><td>UDFs (continued)</td><td>
<table>
<tr><th>To do the following...</th><th>You must have...</th></tr>
<tr><td>replace an existing function</td><td>the DROP FUNCTION privilege on the function or on the database containing the function.</td></tr>
</table>
<p>When a system is first initialized, only user DBC can grant these new privileges with the WITH GRANT OPTION. In order for you to pass on the CREATE FUNCTION privilege, DBC would have to grant you the CREATE FUNCTION privilege with the WITH GRANT OPTION.</p>
<p>When new users are created, they do not get the option to create functions by default. You must explicitly grant that option. If you grant users the CREATE FUNCTION privilege on a database, any function the users creates automatically has DROP FUNCTION, EXECUTE FUNCTION, and WITH GRANT OPTION on the function.</p>
<p>To grant execute privilege to all functions in a database to a particular user, use the following:</p>
<pre>GRANT EXECUTE FUNCTION ON SYSLIB TO User_xyz;</pre>
<p>To grant execute of just a single function to a user, use the following:</p>
<pre>GRANT EXECUTE ON SPECIFIC FUNCTION SYSLIB.sales TO User_xyz;</pre>
<p>For syntax on granting privileges, see <i>SQL Reference: Data Definition Statements</i>.</p>
</td></tr>
<tr><td>User-defined Methods (UDMs)</td><td>privileges required for creating and using UDMs are the same as UDTs. See "User-defined Types (UDTs)" below.</td></tr>
<tr><td>User-defined Types (UDTs)</td><td>
<p>UDTUSAGE, UDTTYPE, or UDTMETHOD privileges.</p>
<table>
<tr><th>To...</th><th>You must at least have the following privilege...</th></tr>
<tr><td>use a UDT in a table or view and to execute all methods associated with that UDT</td><td><p>UDTUSAGE. You can grant this privilege at both the database and object level. It is not an automatic privilege and a user must be granted this privilege or acquire it through a role. A user granted UDTUSAGE WITH GRANT option can grant it (optionally also with the WITH GRANT option) to others.</p><p>UDTUSAGE allows users to execute all SQL statements that reference existing UDTs and their existing methods. It does not permit creation of new UDTs, altering the ordering, casting, or transform behavior of existing UDTs, or creating new methods.</p></td></tr>
<tr><td>create, alter, or drop a UDT</td><td><p>UDTTYPE is only granted at the database level. This privilege includes all the abilities of UDTUSAGE plus the ability to create, alter, and drop UDTs. Users can also create or drop cast, ordering, or transform properties.</p><p>Users with this privilege cannot, however, create new methods or drop and replace existing methods.</p></td></tr>
<tr><td>create new methods or drop and replace existing methods</td><td>UDTMETHOD. This privilege includes all the abilities of UDTTYPE plus ability to use, create, drop or alter any UDT and its methods without any restrictions.</td></tr>
</table>
</td></tr>
</table>

Granting Privileges

You can use the GRANT statement to:

- explicitly give users, databases, roles, and PUBLIC privileges on a database object
- grant a role to a user or another role.

The following table summarizes the privileges, authority, and requirements for a recipient (grantee) of a new privilege.

Recipient	Privilege Acquisition and Type
Creators	Some general usage notes include: • The creator of a user or database needs to submit an explicit GRANT CREATE statement to grant to that new user the explicit privilege to create functions, databases, users, or procedures in the space that the user owns. Also, to create roles and profiles, the user or database must be granted the privilege to do so. Note that roles and profiles are not owned. • The creator or owner of a table or view needs to submit an explicit GRANT statement to grant to other users the SELECT privilege or any other privilege on that table or view. • (The creator has the EXECUTE privilege to statements in the body of the created macro or stored procedure. The system checks that privileges to the objects targeted by those statements are performed against the immediately owing user or database.) • To grant the EXECUTE PROCEDURE privilege to other users, the creator must first have the EXECUTE PROCEDURE privilege with the WITH GRANT OPTION. (Also see "Limiting Data Access with Stored Procedures" on page 199.) • To grant the EXECUTE FUNCTION privilege to other users, the creator must first have the EXECUTE FUNCTION privilege with the WITH GRANT OPTION.
Owners	Owners have the implicit privilege to grant explicit privileges on their owned objects (except for stored procedures and functions). For complete details, see "Stored Procedures" in *SQL Reference: Stored Procedures and Embedded SQL*. For details on user-defined functions, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.)
Users	Although a user or database has certain explicit privileges automatically granted on itself when it is created, a new user/database must be explicitly granted others. For a complete list, see "Privileges That Must Be Explicitly Granted" on page 165. For example, before Jones can create a new database in Jones, the DBA or the creator of Jones must submit a GRANT statement to explicitly grant to Jones the explicit CREATE DATABASE privilege. • To grant the CREATE privilege on objects to other users, the creator must first be granted CREATE privilege … WITH GRANT OPTION. For more information, see "Creating a Database or User" on page 45. • To grant a role to another user, a user must first be granted the role with the WITH ADMIN OPTION. For more information, see "Using Roles to Manage Privileges" on page 134. • To grant the EXECUTE PROCEDURE privilege to other users, the creator must first be granted the EXECUTE PROCEDURE privilege with the WITH GRANT OPTION. For more information, see "Limiting Data Access with Stored Procedures" on page 199.

You can use the Teradata Administrator to grant or revoke access, object, system, column or logon rights. For more information, see "Setting Access Rights" in *Teradata Administrator User Guide*.

Granting the Ability to Grant Privileges

The WITH GRANT OPTION statement gives the recipient "Grant Authority." The grantee holding this authority may then grant the privilege to other users or databases. The recipient of an explicit privilege may be as described in the following table.

Recipient of WITH GRANT OPTION	Description
username or *databasename*	The individual user or users named. Up to 25 can be specified.
All *username*	The named user and all descendants in the hierarchy. If username is DBC, then the statement is translated internally to PUBLIC.
PUBLIC	Every user and database inherits explicit privileges held by PUBLIC.

The following table describes some examples of granting and revoking.

Action	Description
Granting a role to a user or another role	You must have the WITH ADMIN OPTION on the role and you must have: • The privilege itself and have the WITH GRANT OPTION on the privilege. • That privilege on, or be an owner of, the same object you are granting the privilege on. For more information, see "Using Roles to Manage Privileges" on page 134. For a complete discussion on the WITH GRANT option, see *Security Administration*.
Granting privileges to every user in the database	Use the PUBLIC keyword to grant a privilege to all users. When you grant one or more explicit privilege on an object to PUBLIC, one row (indicating the public-object pair) is inserted in the DBC.AccessRights table for each granted privilege.
Revoking Granted Privileges from all users	Revoke one or more explicit privilege from PUBLIC. However, you cannot selectively revoke a privilege from an individual user that is inherited from PUBLIC. **Note:** The statement REVOKE … FROM ALL DBC is not translated to REVOKE … FROM PUBLIC. The ALL DBC form of REVOKE is used only to delete DBC.AccessRights rows for public rights granted in pre-V2R5.0 releases of Teradata Database.

Although a user can have his explicit privileges on objects he owns revoked, this simply causes him not to be able to directly access the objects. He can always explicitly grant those explicit privileges back any time since he has implicit privileges to do so.

For example, if a user has lost explicit privileges on tables, views, or macros he owns, the system may grant him the ability to still access them because the owner has implicit privileges on tables, views, and macros. He can also explicitly grant the privilege back to himself by submitting the appropriate GRANT statement.

Using the GRANT *privilege* Statement

The following table describes some of the privileges you might use to grant privileges to other users or roles. For example, to grant the privilege to alter functions, submit the following SQL statement:

```
GRANT ALTER FUNCTION ON object TO grantee;
```

For a full list of privileges, see "GRANT (SQL Form)" in *SQL Reference: Data Definition Statements*.

Privilege	Purpose
ALTER FUNCTION	Recompiles a user-defined function and allows you to switch the protection modes on functions.
ALTER PROCEDURE ALTER EXTERNAL PROCEDURE	Recompiles a stored procedure or external stored procedures. See *SQL Reference: Data Definition Statements* for syntax.
CHECKPOINT	Creates a synchronization entry in a journal table or an index analysis workload. Checkpoints are used in ARC and load client utilities such as TPump, FastLoad, and MultiLoad.
CREATE…	Creates a database, user, role, profile, table, view, macro, stored procedure, permanent journal table, index (secondary), join index, hash index, trigger, or user-defined function.
DELETE	Deletes rows from a table.
DROP…	Remove a database, user, role, profile, table, view, macro, journal table, stored procedure, index (secondary), join index, hash index, trigger or user-defined function. DROP TABLE also allows ALTER TABLE, creating an index on a table, collecting statistics on a table, and modifying a database or user.
DUMP	Archive an AMP, AMP range, or AMP cluster, or one, several, or a range of databases, data tables, journal tables, or users. Used in ARC as well as online archiving.
EXECUTE	Execute a macro.
EXECUTE FUNCTION	Execute a UDF. For more information, see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.
EXECUTE PROCEDURE	Refers to the corresponding CALL statement.

Privilege	Purpose
INDEX (table level only)	Privilege that includes the: • CREATE INDEX and DROP INDEX privileges • COLLECT STATISTICS and DROP STATISTICS privileges
INSERT	Load new rows in a table, directly or through a view.
REPLCONTROL	To define and alter replication groups, you must first have the REPLCONTROL privilege. This privilege is similar to the ROLE and PROFILE privileges in that it cannot be granted to specific tables, databases, or users.
RESTORE	Used in ARC to: • Restore by AMP, database or user, journal or table. • or execute the following: • DELETE JOURNAL to drop a permanent journal. • ROLLBACK to use a before-image permanent journal to restore tables (than write to that journal) to their state before a modification. • ROLLFORWARD — Uses an after-image permanent journal to update tables (that write to that journal) to reflect a modification.
SELECT	Select the information in one, many, or all columns from a table or view.
UDTMETHOD	Allows you to use, create, alter, and drop UDTs as well as create new methods or drop and replace existing methods.
UDTTYPE	Create or drop UDTs. It also permits you to alter the ordering, casting, or transform behavior by referencing pre-existing methods.
UDTUSAGE	Execute all SQL statements that reference existing UDTs and their existing methods.
UPDATE	Modify column values in a table, directly or through a view.

Granting Logon Privileges

The following conditions should exist to use the GRANT LOGON and REVOKE LOGON statements:

- The DIPACC script has been run to create the special security macros DBC.LogonRule and DBC.AccLogRule.
- A Teradata Database security administrator user has been created; for example, username SecAdmin.
- User SecAdmin has been granted the EXECUTE privilege on DBC.LogonRule

If these conditions exist, the security administrator can execute the GRANT LOGON or REVOKE LOGON statements any time after installation to add or remove user names on individual host connections as needed.

Statement	Comments
GRANT LOGON	Gives users permission to log on to the Teradata Database from specific client systems using a pre-validated logon request. To execute a GRANT LOGON statement, you must hold execute privileges on the macro DBC.LogonRule.
REVOKE LOGON	Retracts permission to log on to the Teradata Database from specific client systems. After installation, use the REVOKE LOGON statement to change the system default by first removing logon privileges from all users from all hosts. Then, you can submit the GRANT LOGON statement to assign individual users to specific host IDs.

To change the system default:

1. Submit the REVOKE LOGON statement to remove logon privileges from all users from all hosts.
2. Submit the GRANT LOGON statement to assign individual users to specific host IDs.

The GRANT LOGON and REVOKE LOGON statements store rows in the DBC.LogonRuleTbl.

You can use the Teradata Administrator to grant logon privileges. For more information, see “Grant or Revoke Logon Rights” in *Teradata Administrator User Guide*.

Managing the DBC.AccessRights Table

The system uses DBC.AccessRights to validate that privileges exist before permitting actions to occur on database objects. As permissions change or new objects are created, the DBC.AccessRights grows. This table can become very large. Do the following to reduce the size of DBC.AccessRights:

- Use roles and PUBLIC.
- Revoke object level privileges when there is database privilege for a containing database.
- Grant privileges at the database level instead of at the object (table, view, macro, and so on) level.

It is good to minimize I/Os to DBC.AccessRights because when a user submits a statement during a session, the following validation for privileges occurs:

1. The system searches the cache or DBC.AccessRights for a userID-objectID pair entry for the required privilege. If WITH GRANT OPTION is needed, the system validates the privilege for this pair to see if user has this option.
2. The system continues validating privileges by checking for the following:

IF the…	THEN the system searches the …
entry is not found and user has a current role (that is not set to ALL)	cache or DBC.AccessRights for roleID-objectID pair entry for the required privilege. Go to step 3.
current role of the user is ALL, for each direct and nested role	cache or DBC.AccessRights for roleID-objectID pair entry for the required privilege. Go to step 5.

3. If the entry is still not found, the system retrieves from the cache or DBC.RoleGrants all roles nested within the current role.
4. For each nested role, the system searches the cache or DBC.AccessRights for roleID-objectID pair entry for the required privilege.
5. If still not found, the system searches the cache or DBC.AccessRights for a public userID-objectID pair entry for the required privilege.
6. If still not found, the system checks for ownership.

   If owner privileges exist for the user, the system then checks if granting valid explicit privilege is permitted.

Revoking Privileges

Use the REVOKE statement to take away (from a user, database, role, or PUBLIC) an explicit privilege on a database, user, role, profile, table, view, macro, stored procedure, or user-defined function.

To revoke a privilege, you must either:

- Be an owner of the object.
- Have the same explicit privilege you are revoking. Or have WITH GRANT OPTION if you are not revoking from yourself.

You can use the Teradata Administrator to revoke privileges. For more information, see “Setting Access Rights” in *Teradata Administrator User Guide*.

Effects on DBC.AccessRights System Table

When revoking privileges, the system checks for the following rows to remove from the DBC.AccessRights table:

- Automatically inserted rows generated from a CREATE statement.
- Explicitly inserted rows from a GRANT statement.
- If revoking an explicit privilege to an object from PUBLIC, one row in DBC.AccessRights for each specified privilege.

  Note: All users are equal members of PUBLIC; no one can be selectively denied an explicit privilege held by PUBLIC. If you want to restrict only certain or specific users, try using roles instead.
  Once an explicit privilege on an object is revoked from PUBLIC, only an owner (for some privileges), or a user, database, or role that has been or is later individually granted the privilege, can perform that function on the object. (Also, for some privileges, an owner can re-grant itself that explicit privilege.)

The REVOKE statement can remove privileges (and thus rows from DBC.AccessRights table) for the object-privilege pairs.

IF you REVOKE FROM …	THEN the specified privileges are removed from …
`ALL DBC`	every user in Teradata Database. **Warning: On pre-V2R5.0 releases, using the REVOKE…FROM ALL DBC statement on DBC tables also removes the privileges from user DBC. If this occurs, privileges for user DBC can only be restored with the assistance of the Teradata Support Center.** On pre-V2R5.0 releases, a grant to PUBLIC caused one object-privilege row to be inserted for every user in the system. Use REVOKE…FROM PUBLIC (instead of REVOKE…ALL DBC) to remove these rows from DBC.AccessRights after upgrading to V2R5.0 or later. For more information, see “Revoking Privileges from PUBLIC or ALL DBC” in *Security Administration*.

IF you REVOKE FROM ...	THEN the specified privileges are removed from ...
`ALL` *`username`*	the specified user and all its descendents in the hierarchy.
`PUBLIC`	username PUBLIC, which has one row per object-privilege pair.

`username`
`or` *`databasename`*
	the specified user or database.

`rolename`	the specified rolename. The rows for that rolename in DBC.AcessRights are removed.

It is important to note that the REVOKE statement:

- Is not automatically issued for privileges granted by a grantor dropped from the system.
- Does not cascade through the hierarchy unless you specify ALL.
- Can remove the ADMIN OPTION from the creator of a role, so thereafter the creator will not be allowed to grant, revoke, or drop the role the creator created.
- Cannot be used to revoke implicit privilege due to ownership.
- Removes, for a privilege revoked at the database or user level, all rows of that privilege for all objects within the database or user.

  Note: A GRANT statement issued for a database or user does not insert rows in DBC.AccessRights for the individual tables, views, macros, stored procedures, user-defined functions, database, or users below that database or user in the hierarchy. Therefore, you cannot revoke specific privileges at the object level for any specific table, view, macro, stored procedure, user-defined function, database or user.

For a complete discussion of privileges and the ramifications of using GRANT, WITH GRANT OPTION, WITH ADMIN OPTION, and REVOKE statements, see *Security Administration*.

Revoking Privileges from PUBLIC or ALL DBC

Because the method of how privileges granted to PUBLIC changed during V2R5.0, which REVOKE statement to use depends on when the privilege was originally granted. To revoke PUBLIC privileges properly, use the procedure in the following table.

IF privileges were granted on ...	IN this release...	THEN use the following statement...
either a non-DBC or DBC table granted to PUBLIC or ALL DBC[a]	V2R5.0 or later	`REVOKE <`*`privilege`*`> FROM PUBLIC;`
a *non-DBC table* granted to PUBLIC or ALL DBC	before V2R5.0	`REVOKE <privilege> FROM ALL DBC;`

IF privileges were granted on ...	IN this release...	THEN use the following statement...
a *DBC table* granted to PUBLIC or ALL DBC[b]	before V2R5.0	Use one of the following: • *REVOKE <privilege> FROM <user>;* • *REVOKE <privilege> FROM <list of users>;* • *REVOKE <privilege> FROM ALL <non-DBC user>;* Revoking from ALL <user> means that the specified user and all its descendents will no longer have that privilege. Do not revoke from ALL DBC. **Warning: When revoking privileges on DBC tables, the REVOKE...FROM ALL DBC statement will also remove user DBC privileges on these tables! If this occurs, only TSC can help re-grant privileges to DBC.** If you need to revoke a privilege on a DBC table granted before V2R5.0 from a very large number of users and you are not able to use *REVOKE...FROM ALL <non-DBC user>*, contact the TSC for assistance.

a. In general, it is best to avoid granting privileges on DBC tables to all users. You should not make DBC tables that widely available and only grant privileges on the DBC tables to system administrators or trusted users.

b. This is for privileges that were not converted over to PUBLIC rights after upgrading from pre-V2R5.0 to V2R5.0 or later. If the conversion was done, however, use REVOKE <*privilege*> FROM PUBLIC;

Granting a Set of Privileges to Multiple Users

Automatically Cascading Privileges

To grant a user or database all privileges on a specific object, use the following statement:

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

The named database or user and every database or user it owns in the future all have privileges on that object.

For example, if you grant explicit privileges to a UserA on its space with the ALL option, then a child user (created below UserA in the hierarchy) is automatically granted those explicit privileges. Also, any new user or databases created below the child of UserA is also automatically granted those privileges. Otherwise, a new user created by a user in its space initially only has automatically granted explicit privileges.

When privileges are automatically granted using ALL, DBC.AccessRights is updated with a row for the privilege-user pair of each privilege. (This applies to users and databases only when they are created. It does not apply when using GIVE.)

Using Roles to Manage Privileges

You can control user privileges at the group level using the following process:

1. Create one or more roles.
2. Grant the appropriate privileges on the appropriate objects to each role.
3. Grant one or more roles to one or more users (enabling each grantee user to acquire the access privileges of the granted role during a session).
4. Define a granted role as the default (or specify ALL) for one or more grantee users.

Any user who logs on with a valid default role has all the privileges that have been granted to that role on the specified objects.

To define a default role for a user, use the CREATE/MODIFY USER statement. For instructions, see "Rules for Using Roles" on page 136.

You can define the default role as ALL. After you submit a CREATE or MODIFY USER statement, setting the default to ALL allows all granted roles to the user. These roles become effective for the user when the newly created or modified user logs on.

Note: A role cannot be granted to a database.

Use SET ROLE ALL to allow privilege validation through all the roles of a user, both directly-granted and nested. To enable all roles for a user within one session, use one of the statements described in the following table.

Syntax	Comment
`SET ROLE ALL;`	A user may set the current session to ALL even if the user does not have any roles granted.
`CREATE USER...DEFAULT ROLE = ALL;`	The creator does not have to be granted any roles. The default setting becomes effective for the user when the newly created user logs on.
`MODIFY USER...DEFAULT ROLE = ALL;`	The user being modified does not have to be granted any roles. The default setting becomes effective for the user when the modified user logs on.

If the user has not already been granted any roles, then any roles granted to the user after the SET ROLE ALL statement has been submitted are enabled for privilege validation in that session.

If the role of a user is already set to ALL (either by default or by means of the SET ROLE ALL statement), any roles granted to a user are effective immediately.

Setting the default for a user does not affect the role setting for a user until the next time they log on.

Note: SET ROLE is not supported within a stored procedure definition.

Effect of the GIVE Statement on Privileges

When you give a user to another immediate owner, explicit privileges are not altered. The GIVE statement does not alter DBC.AccessRights. The database or user that you GIVE does not receive any privileges from its new immediate owner.

The new owners gain only *implicit* privileges over the transferred object and the old owners lose them. The ALL option does not apply.

Transferring objects affects both the ownership of space and also the administration of implicit privileges.

For more information on how privileges are affected, see the GIVE Statement under “Data Control Language Syntax” in *SQL Reference: Data Definition Statements*.

For more information on transferring ownership, see “Changing the Hierarchy with GIVE” on page 33 and “Increasing Space by Giving Ownership” on page 107.

CHAPTER 7 Controlling and Tracking Access to the Database

This chapter introduces some security issues. For full discussion on topics such as setting up authentication mechanisms, network logon string parameters, and the Teradata Database Generic Security Services library, see *Security Administration*.

There is no standard level of auditing for all administrative users to follow because this varies from site to site based on specific business requirements. While Teradata makes some recommendations, you will need to customize the suggestions to your own environment.

Best Practices for Security

Some of the best practices for security overlap with the best practices for setting up users and roles: see "Best Practices for Setting Up Users and Roles" on page 125. Consider the following practices for good security:

- Create a *separate* 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 administrative user should have access to something they do not need to access.
- Ensure that all users are uniquely identified.

  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 user account, each user is held accountable for their specific actions.
- Use roles to control and manage privileges according to job description or responsibility.

  Make all users belong to a ROLE and inherit its explicit privileges. Because it is easier to manage the privileges of a few roles rather than for each individual user in the entire system, there is less chance of improper assignments or failure to grant privileges.
- Grant only the minimum number of privileges required for a job. In other words, do not grant a user privileges they do not need.
- Use strong password controls.

  Strong password controls means using a combination of alpha, numeric, and special characters, not allowing the use of username as apart of the password, expiring passwords at least every 90 days, and restricting re-use of passwords for at least 270 days.

Additional security measures to consider include:

- Use profiles to manage specific password requirements for different sets of users.
- Give users access to databases that contain only views, macros, stored procedures, or functions and not the tables containing the actual data.

  Make sure users do not have direct access to data tables unless they are performing batch operations.
- Enable Teradata Database access logging or query logging to provide an audit trail of database events that can be used to detect security violations.

  Enabling access logging can help track necessary actions and changes for rollback or reassignment of privileges.
- Enable database query logging (DBQL) if your auditing needs require it.

  While access logging tells you when a user has tried to access data objects, DBQL can log the actual SQL executed by a user. In addition, query logging provides better performance than access logging. (See Chapter 13: "Tracking Processing Behavior with the Database Query Log (DBQL).")
- Establish procedures for regular monitoring of all audit logs.
  - Regularly monitor audit logs to ensure that users are only performing activities that they have been explicitly authorized to do.
  - Determine the frequency of audit log reviews based on risk factors such as how critical the data warehouse is to business operations, the value, the sensitivity or criticality of the information stored, the past experience of system compromise or misuse, and the extent to which the system is accessible through non-secure networks.
- Periodically archive audit logs for use in security investigations and access control monitoring.
- Ensure that attempts to access data by unauthorized users are automatically checked and prevented at several levels (from client machine to data table).

Example of Shared Accounts

Teradata recommends that you do not allow users to access the system using shared accounts or to remotely connect to the client using a shared account.

Among other information, the LogonSource column of the DBC.LogOnOff view displays the IP address and the OS logon user account for the user that logged onto the database. If you allow users to share a user account and submit the following query:

```
SELECT LogonDate, LogonTime, LogonSource
FROM DBC.LogOnOff
WHERE UserName = 'DBADMIN' AND Event = 'Logon'
ORDER BY 2, 3
;
```

Teradata Database might return a report similar to the following:

```
LogonDate   LogonTime LogonSource
--------- ----------- -----------------------------------------------------
 07/06/28 08:42:06.13
 07/07/30 11:43:21.00 (TCP/IP) 1191 141.206.35.252 DBC      4612  ADMINISTRATOR
 07/07/30 11:55:30.59 (TCP/IP) 11A2 141.206.35.252 DBC      4164  ADMINISTRATOR
```

```
07/06/25 12:22:42.91
07/06/25 12:22:47.94
07/07/30 12:27:37.95 (TCP/IP) 11C0 141.206.35.252 DBC      4164  ADMINISTRATOR
07/06/25 12:34:27.11 (TCP/IP) 0DBA 141.206.35.252 DEVBOX   712   ADMINISTRATOR
07/07/30 12:53:35.37 (TCP/IP) 11CB 141.206.35.252 DBC      4164  ADMINISTRATOR
07/07/30 12:53:54.52 (TCP/IP) 11CC 141.206.35.252 DBC      4164  ADMINISTRATOR
07/07/30 13:01:18.71 (TCP/IP) 11CF 141.206.35.252 DBC      4164  ADMINISTRATOR
07/07/31 14:14:42.81 (TCP/IP) 0533 141.206.35.252 DBC      4252  ADMINISTRATOR
07/08/01 15:36:01.34 (TCP/IP) 0839 141.206.35.252 DBC      1136  ADMINISTRATOR
07/07/31 16:30:32.77 (TCP/IP) 0588 141.206.35.252 DBC      5620  ADMINISTRATOR
```

The LogonSource column repeats the same IP address even despite different users logging in because they all logged in to the client remotely as ADMINISTRATOR.

A security administrator or auditor would have to try correlating the information in the DBC.LogOnOff log with the Security Event log for the system used for the connection. The Security Event log on your system shows each Logon event including the "Source Network Address" (that is, the IP address) of the PC used for the logon. This can be used to trace the logon event back to a client system for an end user. However, you can avoid this by following the best practice of making sure all users are uniquely identified on the system and that they do not use shared accounts.

In addition, configure your systems to only allow logons of privileged user (such as the database administrator user or root) through a local console connection. Restrict shared user IDs and allow only one concurrent session at a given time.

Following these practices helps ensure that all of the security logs can be properly correlated if there is some investigation needed of a security incident.

Controlling Security for Users Authenticated by the Database

Users authenticated by the Teradata Database must supply a valid username and password directly to the database server upon logon. You can select the authentication method through the Logon Information screen of Teradata Manager or select the authentication method through the Gateway Global utility.

The following section discusses how to secure your database by setting up password controls.

Securing Access with Passwords

For users to establish a session on the Teradata Database system, they must enter a username and password at logon. Upon successful logon, the username is associated with a unique session number until the user logs off.

When you create a new user, you also create a temporary password for the user. If the DBC.Users.PasswordChgDate value is set to 0, the user will be prompted and required to set a new password upon the immediately next logon attempt. You cannot submit a CREATE USER statement without defining a password (although under certain conditions you can immediately modify the user for a null password, as explained in "Password Security for Users" on page 131).

Caution: No users should ever write down passwords or share them among other users.

The system validates passwords during a CREATE USER statement for a new user, during a MODIFY USER statement for an established user, or during logon for an expired password.

Customizing Your Password Controls

To see the current password security defaults, select all columns of the system view DBC.SecurityDefaults which shows the settings in the DBC.SysSecDefaults table:

```
SELECT * FROM DBC.SecurityDefaults;
```

To change any of the settings, submit an UPDATE statement to change a column defining the password control. For example, to set the minimum number of characters allowed in a password to 8, submit the following:

```
UPDDATE DBC.SecurityDefaults SET PasswordMinChar = 8 ;
```

You can update other fields such as Password Digits which specifies whether numeric characters are required for a password. Additional password options include:

- whether to require or disallow a mixture of upper and lower case characters, special, numeric or alpha characters (PasswordSpecChar)
- number of days before passwords expire (ExpirePassword)
- amount of time to elapse before an expired password can be re-used (PasswordReuse)
- the minimum and maximum length of a password string (PasswordMinChar and PasswordMaxChar)
- the number of unsuccessful logon attempts before a user is locked out and the amount of time to elapse before unlocking the user (LockedUserExpire)
- whether to prohibit passwords from containing the DBS username or from containing certain restricted words. (PasswordRestrictWords)

The initial defaults are as follows:

- Do not expire passwords
- Do not lock out a user on an erroneous password
- Allow from 1-30 characters that can include digits and special characters
- Allow unlimited logon attempts
- Allow immediate password reuse
- Passwords are case insensitive
- Username is allowed in password string
- Do not prohibit passwords from using words found in the restricted words list[1] of DBC.PasswordRestrictions

Note: Starting with Release 6.2, passwords were automatically converted to Unicode and shareable among character sets. However, password control functions could not be shared.

[1] See “Adding and Removing Restricted Words” in *Security Administration* for information on customizing which words to include in the restricted words list. See “Using Restricted Words to Control Password Content” of *Security Administration* for a list of default words (approximately 2000 words and 400 personal names in English) included in the DBC.PasswordRestrictions table.

Starting Teradata Database 12.0 and forward, password controls are also shareable between character sets.

These parameters are set at the system level and apply to all users with the exceptions described in the following table.

IF a user...	THEN ...
was created with a NULL password	the user is not affected by password settings.
belongs to a profile	the profile specification takes precedence; if NULL, the value in DBC.SysSecDefaults takes precedence.
is set up for external authentication	external authentication takes precedence, regardless of the client from which the logon originates.

An advantage for setting password attributes at the profile level is that when you change a password control setting, it does not require a system restart. The new setting will take effect on the next user logon. For more information, see "Using Profiles to Manage Settings" on page 140 and "Setting Password Controls" in *Security Administration*.

Handling Forgotten Passwords

If a user forgets his password, or you set a maximum for erroneous logon attempts and a valid user becomes locked out, submit a MODIFY USER or MODIFY PROFILE statement with the RELEASE PASSWORD LOCK option. You can assign a new temporary password, which the user can change during the session.

Note: When an administrative user resets a password for another user, the DBC.OldPasswords table is *not* updated and the password string is not verified with password reuse rules. However, it is updated when the user changes his own password.

For information on creating a temporary password for the first login of a pre-existing user, see "Procedure for Creating a Temporary Password" in *SQL Reference: Data Definition Statements*.

Caution: Do not lose the password for user DBC: user DBC could be locked out (because of the MaxLogonAttempts attribute) and only user DBC can modify user DBC! If this happens, contact Teradata Support Center and they can unlock DBC using the TSTSQL console.

Tracking Changes to Passwords

Users authenticated by the database can modify their passwords without special privileges. The DBC.DBase table stores the date and time a password is changed by a user.

Query the DBC.Users view (which references the DBC.DBase table along with some other system tables), selecting the columns PasswordLastModDate and PasswordLastModTime, to see the latest activities against passwords.

For example:

```
SELECT UserName, PasswordLastModDate, PasswordLastModTime FROM
DBC.Users;

 *** Query completed. 6 total rows found. 3 columns returned.
 *** Total elapsed time was 1 second.

UserName                       PasswordLastModDate  PasswordLastModTime
------------------------------ -------------------  -------------------
DBC                                       07/06/25             12:15:27
TDPUSER                                   07/06/25             12:24:25
SystemFe                                  07/06/25             12:24:31
SysAdmin                                  07/06/25             12:24:31
Crashdumps                                07/06/25             12:29:01
Sys_Calendar                              07/06/25             12:29:0
```

Controlling Security for Users Externally Authenticated and Authorized

Teradata Database supports three methods of externally authenticating users.

Authentication Method	Description
Single Sign-on (NTLM and KRB5)	Teradata Database verifies the user through his or her initial network logon rather than require additional username and password for logging on to the database.
Directory server (LDAP)	This authentication method is for sites that employ directory servers. In particular, sites that employ a single centralized repository to manage users for multiple systems with multiple application accounts.
Extensible user	Customer sites may use custom authentication to access the system. This topic is not covered in this book.

Single Sign-On

Using Single Sign-on (SSO) allows users on Windows clients[2] to access a Teradata Database system[3] based on their authenticated network usernames and passwords. This simplifies the procedure that requires users to enter an additional username and password when logging on to Teradata Database through client applications.

Single Sign-on must be enabled for the Teradata Database configuration in the DBS Control and the Gateway Control record. The initial default is enabled (ON). For details, see “Network

[2] A Windows client is required for Kerberos authentication. Query Director, if used, must also run on Windows.

[3] Windows clients can use Kerberos to log on to Windows, UNIX MP-RAS and SUSE Linux Enterprise Server 10 Teradata servers. However, those Windows clients must be running at Teradata Database 12.0 also.

Logon Formats" in *Security Administration* and "SET EXTAUTH" in *Graphical User Interfaces: Database Window and Teradata MultiTool.*

In order for existing TDPs that use implicit logon protocol to function properly, leave the default value of 0 (ON) unchanged. ON ensures that both external authentication and traditional logons are accepted; any other value rejects one form of logon.

To employ SSO, client users need to properly set up the Teradata Database interfaces and applications, such as ODBC or JDBC for Teradata Database, Teradata Manager, Teradata load utilities, and so forth, as instructed in the relevant client documentation.

In general, the procedure for setting SSO on Teradata Database is as follows:

1 Log on to the Teradata Database system as user DBC.

2 Make sure the DBC.AccLogRule macro exists. If it does not, create it as follows:

   a In the Database Window, access the Supvr icon and start the DIP utility:

   ```
   start dip
   ```

   b Go to the supervisor window indicated and log on as user DBC.

   c Select the option for the DIPACC (Access Logging) script.

3 Start BTEQ and submit a GRANT statement that grants to your database administrative user (for example, DBADMIN) the EXECUTE privilege on macro DBC.AccLogRule.

4 Log off as DBC and log on again as the user with GRANT LOGON privilege (for example, DBADMIN).

5 Determine whether every Teradata Database user name is unique.

IF you ...	THEN you ...

can guarantee that every username will always be unique across all domains	• Can issue a GRANT LOGON statement to each existing user who will be logging on with external authentication. For example:
	`GRANT LOGON ON ALL TO user1 WITH NULL PASSWORD;`
	• Can create new users using the form *username*, followed by GRANT LOGON ON ... WITH NULL PASSWORD. For example:
	`CREATE USER newuser2 AS PERM=500000,`
	`PASSWORD=Jim2;`
	`GRANT LOGON ON ALL TO newuser2 WITH NULL`
	`PASSWORD;`

cannot guarantee that every username will always be unique across all domains when using NTLM, Kerberos, or LDAP where the LDAP directory is Active Directory and the server is running on Windows	• Must assign host (domain) names to network client groups, and associate the appropriate users with each domain. (For instructions, see "Logon Controls" in *Security Administration.*)
	• Must append a domain name to each new user name with an @ sign. Enclose the string in straight quotes as in "username@domainname". For example:
	`CREATE USER "Bob@esdev3" AS PERM=1000,`
	`PASSWORD=Bob3;GRANT LOGON ON ALL TO "Bob@esdev3"`
	`WITH NULLPASSWORD;`
	Note: Support for user names that are not unique across all domains is deprecated and will be removed in a future release.

6 If you plan to use domains despite the fact that support is deprecated, and have existing users whose names must be converted, follow the procedure below. This procedure converts your Teradata Database users from the form username to the form *username@domainname.*

a Recreate every existing username to the form "username@domainname", where domainname is the client group name you associated with this user in step 5. Follow each CREATE statement with a GRANT LOGON... WITH NULL PASSWORD statement. Define a separate CREATE/GRANT transaction for each username.

For example:

```
CREATE USER "origusr2@esdev3" AS PERM=1000, PASSWORD=abcd2;GRANT
LOGON ON ALL TO "origusr2@esdev3" WITH NULL PASSWORD;
CREATE USER "origusr3@tsdev3" AS PERM=1000, PASSWORD=efgh3;GRANT
LOGON ON ALL TO "origusr3@tsdev3" WITH NULL PASSWORD;
...
```

b Use GIVE to transfer from each original user to his or her new name the default database and all the objects below it in the hierarchy, as well as all the PERM space allocated to it:

```
GIVE origusr2 TO "origusr2@esdev3";
GIVE origusr3 TO "origusr3@tsdev3";
```

c Drop the old user.

```
DROP USER origusr2;
DROP USER origusr3;
```

7 Log off and quit your session.

8 To enable external authentication in the DBS Control Record, start Teradata Database Window and select the Supvr (Supervisor Window) icon.

9 In the Enter a command subwindow of the Supervisor window, start the DBS Control utility:

```
start dbscontrol
```

The Supervisor window displays:

```
   Started 'DBSCONTROL' in window n.
```

where the number represents the application window in which DBS Control is running.

Note: For details on the Database and Supervisor windows, see *Graphical User Interfaces: Database Window and Teradata MultiTool.*

10 Set the external authentication field to 0 or 2. This field can be set to 0 (the default which is to accept both traditional and external authentication logons), 1 (accept traditional logon but reject external authentication) or 2 (accept only external authentication logons):

```
set ExternalAuthentication ON
```

Note: The DBS Control Record setting for external authentication overrides any Gateway setting.

11 To enable external authentication, start the Teradata Command Prompt window.

12 At the Teradata command prompt, use the gtwcontrol -d command to query the state of the Gateway Control Record fields.

13 Use the gtwcontrol command to control the state of the externalauthentication field as described in the following table.

IF you ...	THEN include...
do not use domain names	only the -a on/only option. For example: `gtwcontrol -a on`
use domain names	both the -g *hostid* and the -a on/only options, where *hostid* is the DBS hostid number displayed in the Configuration utility. For example: `gtwcontrol -g 1 -a on` Enter one gtwcontrol command string for each hostid you created. Or, to turn on external authentication for all host groups, submit gtwcontrol -a on.

14 Use the gtwcontrol command with the -F option to toggle the state of the Append Domain Name field, as described in the following table. However, note that the -F option is deprecated because it will be removed in future releases.

IF your installation ...	THEN the Append Domain Name value must be...
is not set up with domain names	no. **Note:** In this case, a userID with the form "username@domainname" will not be authenticated.
uses domain names as "username@domainname"	yes. However, this value is deprecated in Teradata Database 12.0 and the Append Domain Name value will be permanently set to no in future releases. **Note:** In this case, a userID with the form *username* will not be authenticated.

For more information on using the Gateway control utility (gtwcontrol), see *Utilities*. For information on how to successfully enable Kerberos authentication, considerations for time synchronization between Teradata servers and the Domain Controller, or adding an Active Directory user, see *Security Administration*.

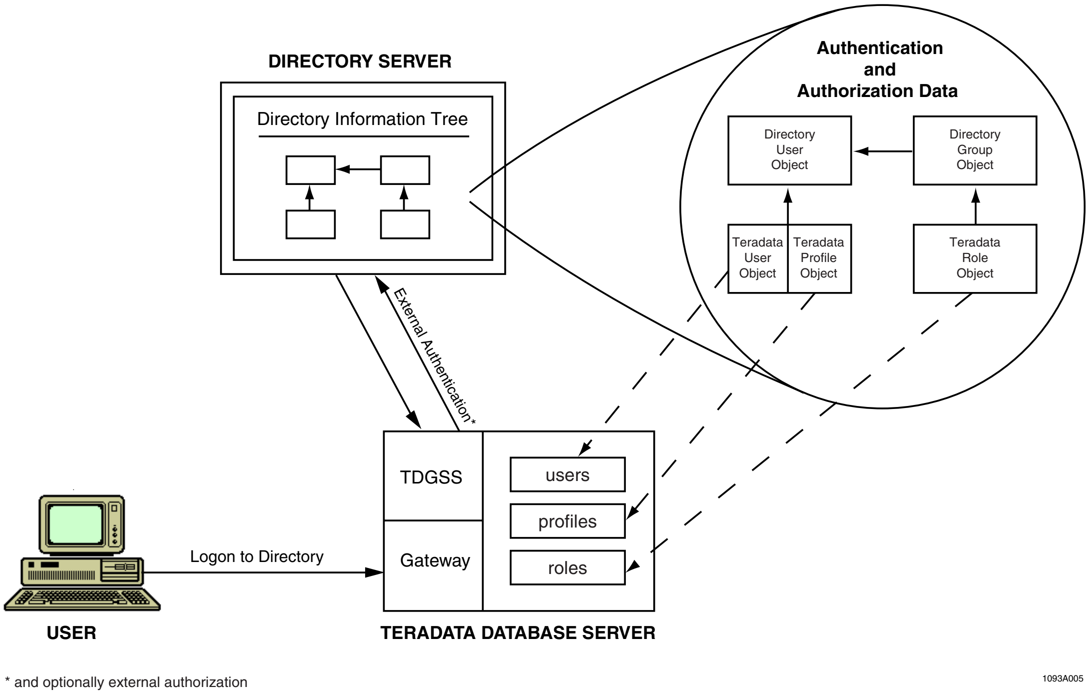
Directory Server Authentication

The following diagram illustrates LDAP authentication. With external authentication using a directory server[4], the user communicates with the Gateway and not directly with the directory server. The authentication and authorization data on the directory server maps directory objects with Teradata objects.

Teradata Database authorizes (allows or denies access to database resources) directory users through privileges of Teradata users or roles they are mapped to. While external

[4] Use a list of server names rather than a single directory so that if the first server becomes inaccessible, the LDAP method would continue to authenticate and authorize users by using the next server named on the list. For more information, see *Security Administration*.

authentication is always required any time a user uses directory services to access data on the Teradata Database, external authorization is optional.



Note: Access to administrator utilities is usually limited to privileges specifically granted in the database. Users of these utilities should log on using their Teradata Database usernames. Users that are externally authenticated (with directory or Kerberos usernames, for example) may not have access to administrator utilities.

A directory user must have at least one mapping to a Teradata user, profile, or role in order to log onto the database. A user without any mappings at all cannot gain access to the database.

If a directory user is mapped to…	Then access to the Teradata Database occurs through…
only EXTUSER	the EXTUSER mapping. The directory user does not need to explicitly specify "user=EXTUSER" in the .logdata command for BTEQ.
one or more roles or profiles but has no Teradata Database user and no EXTUSER mappings	the roles or profiles. The directory user can log in even if no user or EXTUSER mappings are present because of the roles/profiles.
a single Teradata Database user (for example user1) as well as to EXTUSER	the default Teradata Database user (user1) or EXTUSER. See "LDAP Logons" in "Command Line Logons" of *Security Administration.*
multiple Teradata Database users (for example user3 and user4)	mappings to either user3 or user4. The directory user must always specify a "user=" property with the .logdata. This means specifying either "user=user3" or "user=user4."

If a directory user is mapped to...	Then access to the Teradata Database occurs through...
multiple Teradata Database users (for example, user5 and user6) as well as to EXTUSER	user5, user6, or EXTUSER. The directory user must always specify a "user=" property and explicitly list one of the users. This means specifying either "user=user5", "user=user6", or "user=EXTUSER."

A user with no mappings to a Teradata user but has profiles and/or roles is allowed to access the database.

For more information about external authentication and the privileges of externally authenticated users, see *Security Administration*.

Implementing External Authorization

To use external authorization, the security administrator must first set up a directory that is certified by Teradata. This includes Sun Java System Directory Server directory and Active Directory.

You must also install Teradata schema in the directory. If the Teradata directory schema has been installed, the security administrator can add Teradata user, role, and profile objects to the directory server.

For more information on using roles from a directory server, see "Rules for Using Roles from a Directory (External Roles)" on page 137. For more information about using directories and how to set them up, see "Directory-Based Management of Database Users" in *Security Administration*.

Controlling Access to Data

After users are authenticated and authorized, there are still several ways security and access can be restricted. Use any combination of the following to limit access to the data:

- **Privileges** - Control user access to data by granting privileges only to specific users for certain database objects.
- **Views** - Limit user access to table columns or rows that may contain sensitive information.
- **Macros** - Limit the types of actions a user can perform on the columns and rows. Also, you can define which users are granted the EXECUTE privilege on a macro.
- **Stored Procedures** - Like with macros, limit the types of actions a user can perform on the columns and rows. Grant the EXECUTE privilege on stored procedure only to those who should be authorized.

Limiting Data Access with Privileges

An arrangement of privileges control the activities of a user during a session. Privileges are associated with a user, a database, and an object (table, view, macro, stored procedure, join index, trigger, or function), and can be explicitly granted or revoked by an owner of an object, or by a user that has the privilege with the WITH GRANT OPTION privilege. The system verifies user privileges when the user attempts to access an object or execute a function that accesses an object.

Information on explicit privileges is maintained in the system table DBC.AccessRights. You can retrieve this information by querying the view DBC.UserRights (see “Views That Report Explicit Privileges” on page 160). Implicit privileges associated with ownership are not stored in this table (see “Implicit Privileges” on page 162).

Granting Privileges

To grant to another user any privilege on an object the grantor does not own, the grantor must have the explicit privilege along with the WITH GRANT OPTION phrase.

Initially, any owner of a database has complete control over the data in that database. If an owner grants privileges to other users, the owner is sharing control. By granting and revoking privileges, each user can control access to his or her own data. For example, an owner may rescind any explicit privilege on the database that has been granted to one or more users (including the immediate owner) using the REVOKE statement.

The Teradata Database also verifies that the appropriate privileges exist on referenced objects for any user who attempts to access a view or execute a macro or stored procedure. This ensures that a change to a referenced object does not result in a violation of privileges when the view, macro, or procedure object is invoked.

An owner may even give up ownership altogether using the GIVE statement, which relinquishes control of space and its contents to another user or database. In this case, the original owner retains any explicit privileges it has on the space. Unless an owner gives up ownership, he or she retains ultimate control of the database and of users and databases lower in the hierarchy.

Example

Suppose the following occurs:

- The immediate owner of a database allocates space in the database to create user A.
- User A creates a table in the permanent space allocated and grants privileges on the table to user B, WITH GRANT OPTION.
- User B grants the privileges received to user C.

Users can then perform any of the following:

- User A can revoke any privilege that user A has granted to user B, or that user B has granted to user C. Likewise, user B can revoke a privilege user B has granted to user C.
- The immediate owner can drop user A altogether (after first dropping all user A data) to reclaim the permanent space allocated to user A.

User DBC, at the top of the database hierarchy, retains ultimate control of all users and databases.

For more information, see "Overview of Access Privileges" on page 159 or GRANT in *SQL Reference: Data Types and Literals*.

Limiting Data Access with Views

Another method of controlling user access to data is through the use of exclusion views. Because a view executes a SELECT on one or more tables, you can structure views using specific column names in order to exclude columns of sensitive data (for example, salaries or personal information). By using the WHERE clause, you can exclude specific rows.

When you submit the CREATE VIEW statement, Teradata Database:

- Fully expands the view name (resolves the database and table names)
- Verifies that the creating user has the appropriate privilege on the objects being referenced, which can be implicit or explicit privileges

You need to grant the appropriate privileges to users who you want to use the view.

IF you want to ...	THEN you should ...	For more information, see ...
build a virtual firewall between users and the tables they access	structure task-oriented views.	"Isolating Users from the Database Using Views" in *Database Design*
let another user select one or more columns from the view	GRANT the SELECT privilege on the view to that user.	
restrict user access to dictionary data using views	review the All_RI_Child and All_RI_Parent views and system views with the X suffix. Then revoke privileges as needed.	• Chapter 4: "Using Data Dictionary Tables and Views" • *Data Dictionary*

IF you want to ...	THEN you should ...	For more information, see ...
let another user update one or more of the underlying tables through the view	GRANT the UPDATE privilege on the view to that user. **Note:** If an update or delete on a view includes a search condition, which requires read access to find candidate rows, also grant the SELECT privilege on the view.	GRANT (SQL Form) in *SQL Reference: Data Definition Statements*

Example of an Exclusion View

The following statement defines a view that excludes Salary information:

```
CREATE VIEW Employee_Info
AS SELECT EmpNo, Name, JobTitle, DeptNo
FROM Employee;
```

Adding a WHERE clause restricts the view even further to show only data about employees in the Manufacturing department:

```
WHERE DeptNo = 600
```

Although users can only view information for department 600, they can update any row. If you want to show only department 600 and restrict updates or inserts through this view to department 600, you can use the following condition:

```
WHERE DeptNo = 600 WITH CHECK OPTION
```

Updating Tables Through a View

You can grant another user the privilege of selecting or updating, adding, or deleting rows in the underlying table. A possible scenario is as described in the following table.

IF receptionists need to ...	THEN grant them ...
periodically update information about employee address, home telephone, and extension just for their department	Either: • UPDATE privilege on the columns that are allowed to be updated. Such a view also may include WHERE constraints on the data itself. • Both UPDATE and SELECT privileges on another view of the same table that includes names and extensions only for employees in their department. (In this case, the receptionist would have to be careful not to change an employee name with the view.)
access the company telephone extensions of all employees	SELECT privilege on a view that includes only employee name and telephone extension from a table of information about company employees.

For example, if the extension numbers to be updated ranged from 105 to 130 and did not include 123 or 127, the view definition could include the following clause to prevent the receptionist from mistakenly entering an extension for another department:

```
WHERE Extension BETWEEN 105 AND 130
AND Extension NOT IN (123, 127) WITH CHECK OPTION
```

Example of Renaming Table Columns in a View

The following example illustrates a view limiting access to relevant and non-sensitive data and replacing each column name in the table with a title suitable for a report. The GRANT statement gives read access to all users (PUBLIC).

For example, assume you create a new view on Employee called Dept_401 to display non-sensitive data on employees of Department 401 only. The structure might look like this:

```
CREATE VIEW Payroll.Dept_401 AS
SELECT EmpNo (TITLE ’EMPLOYEE//NUMBER’)
,Name (TITLE ’FULL NAME’FORMAT ‘X(35)’)
,HireDate (TITLE ’HIRE//DATE’FORMAT ‘YYYY-MM-DD’)
FROM Payroll.Employee
WHERE DeptNo = 401;
GRANT SELECT ON
Payroll.Dept_401 TO PUBLIC ;
```

When end users access the view, they see what looks like a complete table. They do not know that the underlying base table contains more columns. For example, this query:

```
SELECT * FROM PAYROLL.DEPT_401;
```

returns the following:

```
EMPLOYEE NUMBER       FULL NAME                      HIRE DATE
---------------       ---------------------------     ---------
1006                  Stein, John                    861015
1008                  Kanieski, Carol                870201
1005                  Ryan, Loretta                  861015
1004                  Johnson, Darlene               861015
1007                  Villegas, Arnando              870102
1003                  Trader, James                  860731
```

Using Nested Views

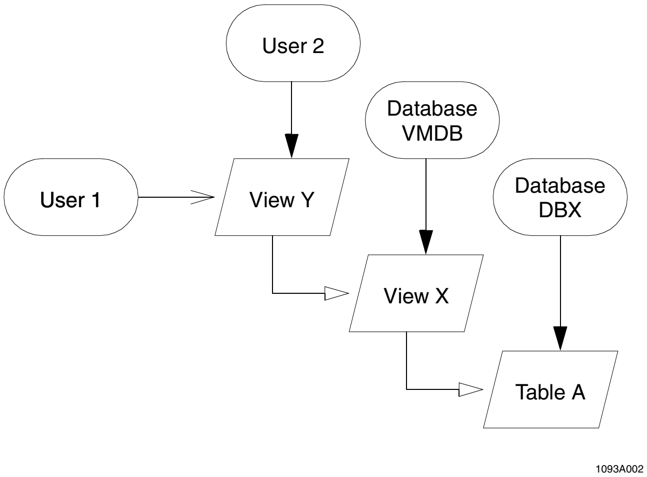
Views that reference other views are called nested views. Access to the underlying objects is as follows:

- Nested views are fully expanded (database and all underlying tables resolved) at creation time.
- Teradata Database validates the nested view privileges of the creator at creation time.
- Teradata Database validates nested view privileges of the immediate owner of the view at execution time.

Nested views are useful for protecting the data in the underlying base table. You can define the WHERE clause to restrict what data a user can view or access. The query must pass each restriction on each view in order to successfully execute.

Example of Nested View Privilege Checking

The following diagram illustrates how privileges are checked when using nested views.



- User 1 accesses View Y.
- User 2 is the immediate owner of View Y.
- Database VMDB owns View X.
- Database DBX is the immediate owner of Table A.

Privileges checked are:

- User 1 privileges on View Y.
- User 2 privileges on View X WITH GRANT OPTION. If User 2 is the same as User1, then the WITH GRANT OPTION is not needed.
- Database VMDB privileges on Table A WITH GRANT OPTION.

If you REVOKE an explicit privilege from any user in the chain, the system issues the following message:

```
3523 An owner referenced by the user does not have [privilege] access to
[databasename.tablename].
```

For more information about views, see:

- *SQL Reference: Fundamentals*
- "CREATE VIEW" in *SQL Reference: Data Definition Statements*
- "User-Restricted Views" on page 118

Limiting Data Access with Macros

Another method of controlling user access to data is through macros. Macros limit the actions a user can perform. The actions can include DDL and DML.

For other users to access a macro, an owner (or someone with the explicit privilege WITH GRANT OPTION) must grant other users the EXECUTE privilege on the macro (or its containing database), plus the appropriate privileges on the target objects.

In addition, the immediate owner of the macro must have the appropriate privileges on the objects referenced by the macro. The WITH GRANT OPTION is also needed for each such privilege if the user associated with the session is not the same as the immediate owner of the macro.

For example, assume a user creates a macro called NewEmp that updates the Personnel.Employee table. If the creator then grants a personnel clerk the EXECUTE privilege on that macro, the clerk can enter new employee data as parameters to the NewEmp MACRO statement rather than using the INSERT statement. Thus, the clerk need not be aware of the database being accessed, the tables affected, or even the result.

You can also use a macro to insert records in an audit trail table to monitor system use, and to record this use for accounting purposes.

Note: A CREATE or REPLACE MACRO statement that contains an EXECUTE MACRO statement succeeds even if the macro in the EXECUTE MACRO statement is not yet defined. Similarly, a CREATE or REPLACE MACRO statement that contains a DDL statement succeeds even if the DDL object is not yet defined.

For security reasons, the CREATE AUTHORIZATION and REPLACE AUTHORIZATION statements are not allowed within a macro.

For more information on macros, see:

- *SQL Reference: Fundamentals*
- *SQL Reference: Data Definition Statements*
- *Data Dictionary*

Limiting Data Access with Stored Procedures

Another method of controlling user access to data is by using stored procedures. Stored procedures limit the actions a user can take on table and view columns and rows. You can create C or C++ stored procedures.

In order to create a stored procedure, you must be explicitly granted the CREATE PROCEDURE privilege on the space in which it will reside (including your own database). You must also have every privilege needed to process the statements and access the target entities in the procedure body. You automatically receive the EXECUTE PROCEDURE and DROP PROCEDURE privileges on any procedure you create.

When a stored procedure is processed, Teradata Database checks the immediate owner of the procedure for access rights on all SQL statements and target objects. If you want another user to execute stored procedures that you create in your own space, you must explicitly grant the EXECUTE PROCEDURE privilege to that user. But, you must already have been explicitly granted every privilege, each including the WITH GRANT OPTION, needed to process the statements and access the target entities in the procedure body.

The rules governing the use of types of SQL statements within a stored procedure depend on whether the user is the immediate owner of the stored procedure being created or executed. The basic rules depend on whether the creator is also the immediate owner.

When the creator is ...	THEN ...
also the immediate owner (the user or database where the procedure was created)	some DDL and DCL statements and dynamic SQL statements are supported within the stored procedure during the procedure creation.
not the immediate owner	DDL, DCL and dynamic SQL statements are not supported within a stored procedure. Specifying such statements results in compilation errors and the stored procedure is not created.

To use stored procedures to control data access, you need to know about:

- Supporting client interfaces and administrative considerations (see "Stored Procedures" on page 73).
- Privilege requirements, statement syntax, rules of creation, use, and execution, control statements, condition handling, and applications (see "SQL Stored Procedures" in *SQL Reference: Stored Procedures and Embedded SQL)*.
- Options and use of the GRANT statement (see "SQL Data Control Language Statement Syntax" in *SQL Reference: Data Definition Statements*).
- Clauses for external stored procedures to directly execute SQL statements through CLIv2 (see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*.)

Tracking Access With Session-Related Views

The session-related system views that you can use to monitor database access include:

- DBC.SessionInfoX
- DBC.LogOnOff
- DBC.LogonRules
- DBC.Software_Event_Log

For more information on all system views, see *Data Dictionary*.

DBC.SessionInfo View

This view provides information about users who are currently logged on, including the session source (host connection, logon name, and application), query band, the current partition, collation, role, password status, type of transaction such as 2PC, LDAP status, and audit trail ID. (For information about collections of sessions initiated under the same logon, use the DISPLAY POOL command; for details, see *Teradata Director Program Reference*.)

If you use a multi-tier client architecture, the LogonSource field of this view can provide distinct source identification as it originated from the server tier, including the user ID and application name.

Note: Data strings for TCP/IP sessions are inserted into the LogonSource field by CLIv2, which truncates strings exceeding 128 bytes.

Example 1

```
Teradata BTEQ 08.00.00.00 for MVS. Enter your logon or BTEQ command:
.logon tdpv/socal
.logon tdpv/socal  Password:
 *** Default Character Set Name 'EBCDIC '
*** Logon successfully completed.
*** Transaction semantics are BTET.
*** Total elapsed time was 0.58 seconds.
BTEQ -- Enter your DBC/SQL request or BTEQ command:
sel logonsource from dbc.sessiontbl;
sel logonsource from dbc.sessiontbl;
*** Query completed. 5 rows found. One column returned.
*** Total elapsed time was 0.44 seconds.
LogonSource
--------------------------------------------------------------------
(TCP/IP) 05BF 155.64.116.42 IETTST      818  DBADMIN  BTEQ  01 LSS
(TCP/IP) 06F9 208.199.59.157 NAG2N2      1396  POINT  BTEQ  01 LSS
TCP/IP) 04DC 10.243.71.25 PW_OLD      2462  ROOT  ARCMAIN  01 LSS
MVS      TDRT    D48734     BATCH    CS210041       SOCAL  BTQMAIN
MVS      TDPV    AN1005     TSO      AN1005         SOCAL  IKJFT01
```

The first three lines report network sessions. The last two lines report channel-connected sessions. The meaning of each field in each type is described in the following table.

Field	Description, Network Sessions	Value in Example 1
First	Connection name or type (literal)	(TCP/IP)
		(TCP/IP)
		(TCP/IP)
Second	Port or socket identifier	05BF
		06F9
		04DC
Third	IP address of the client (host)	155.64.116.42
		208.199.59.157
		10.243.71.25
Fourth	Teradata Director Program Identifier (TDPID)	IETST
		NA2N2
		PW_OLD
Fifth	Client process/thread identifier	818
		1396
		2462
Sixth	Username under which this session logged on	DBADMIN
		POINT
		ROOT

Field	Description, Network Sessions	Value in Example 1
Seventh	Name of the application (executable) under which this session was invoked	BTEQ
		BTEQ
		ARCMAIN
Eighth	Literal, all network sessions.	01
		01
		01
Ninth	Literal, all network sessions.	LSS
		LSS
		LSS

Example 2

Because of the length of LogonSource field, if you combine it with other fields, make use of the BTEQ .SET formatting commands WIDTH and FOLDLINE to avoid data truncation. For example, you can set line width and folding with:

```
 Teradata BTEQ 12.00.00.00 for WIN32.
 Copyright 1984-2007, NCR Corporation. ALL RIGHTS RESERVED.
 Enter your logon or BTEQ command:
.logon sampledb/dbc

.logon sampledb/dbc
Password:

*** Logon successfully completed.
 *** Teradata Database Release is 12.00.00.00
 *** Teradata Database Version is 12.00.00.00
 *** Transaction Semantics are BTET.
 *** Character Set Name is 'ASCII'.

 *** Total elapsed time was 4 seconds.

 *** Total elapsed time was 3 seconds.

 BTEQ -- Enter your DBC/SQL request or BTEQ command:
.set foldline on

.set foldline on
 BTEQ -- Enter your DBC/SQL request or BTEQ command:
.set width 75

.set width 75
 BTEQ -- Enter your DBC/SQL request or BTEQ command:
select logonsource, logondate, logontime from DBC.SessionInfo;

select logonsource, logondate, logontime from DBC.SessionInfo;

 *** Query completed. One row found. 3 columns returned.
 *** Total elapsed time was 2 seconds.

LogonSource
------------------------------------------------------------------------
   LogonDate
   ---------
     LogonTime
   -----------
(TCP/IP) 0DA8 143.206.22.123 SAMPLEDB 698  ADMINISTRATOR BTEQ  01 LSS
    07/10/09
   11:12:53.21
 BTEQ -- Enter your DBC/SQL request or BTEQ command:
```

Example 3

You can narrow the output with the BTEQ SIDETITLE formatting command:

```
Teradata BTEQ 12.00.00.00 for WIN32.
Copyright 1984-2007, NCR Corporation. ALL RIGHTS RESERVED.
Enter your logon or BTEQ command:
.logon tdata/test_user
Password:

*** Teradata Database Release is 12.00.00.00
 *** Teradata Database Version is 12.00.00.00
 *** Transaction Semantics are BTET.
 *** Character Set Name is 'ASCII'.

 *** Total elapsed time was 4 seconds.

BTEQ -- Enter your DBC/SQL request or BTEQ command:
.set foldline on
.set foldline on
BTEQ -- Enter your DBC/SQL request or BTEQ command:
.set sidetitles on
.set sidetitles on
BTEQ -- Enter your DBC/SQL request or BTEQ command:
select * from dbc.sessioninfo;
select * from dbc.sessioninfo;
*** Query completed. 2 rows found. 18 columns returned.
*** Total elapsed time was 2 seconds.
         UserName test_user
      AccountName acct
        SessionNo 1,002
  DefaultDataBase test_user
            IFPNo 16383
        Partition DBC/SQL
    LogicalHostId 1025
           HostNo 1
 CurrentCollation H
        LogonDate 07/10/09
        LogonTime 12:47:40.23
  LogonSequenceNo 00000000
      LogonSource (TCP/IP) 0DA9 141.206.35.252 DBC      2604  ADMINISTRATOR
  ExpiredPassword N
        TwoPCMode N
 Transaction_Mode T
      CurrentRole
      ProfileName
        LogonAcct DBC
             LDAP N
     AuditTrailId DBC
CurIsolationLevel SERIALIZABLE
        QueryBand ?
```

Example 4

The following example shows session information (with sidetitles and foldline set on) for a session that has defined a query band.

```
BTEQ -- Enter your DBC/SQL request or BTEQ command:
sel * from dbc.sessioninfo;

sel * from dbc.sessioninfo;

 *** Query completed. One row found. 23 columns returned.
 *** Total elapsed time was 1 second.

         UserName QBUSER
      AccountName DBC
        SessionNo         1,296
  DefaultDataBase QBUSER
            IFPNo  16383
        Partition DBC/SQL
    LogicalHostId 1025
           HostNo      1
 CurrentCollation H
        LogonDate 07/01/23
        LogonTime 08:47:17.43
  LogonSequenceNo 00000000
```

```
      LogonSource (TCP/IP) 08AD 127.0.0.1 DGUSERTDP 2280  DG120649  BTEQ
  ExpiredPassword N
        TwoPCMode N
 Transaction_Mode T
      CurrentRole
      ProfileName
        LogonAcct DBC
             LDAP N
     AuditTrailId QBUSER
CurIsolationLevel SERIALIZABLE
        QueryBand
 REPJOBID=495;REPORT=Sales;CLIENTMACHINE=XYZ12;REPTYPE=10;

 BTEQ -- Enter your DBC/SQL request or BTEQ command:
```

If no queryband is set, SessionInfo will show a '?' for QueryBand.

DBC.LogOnOff View

The DBC.LogOnOff view provides information about the success and duration of user sessions, in addition to LogonSource information. This view is helpful when you need to know about failed attempts to log on.

For example, the following query returns any failed logon attempts during the last seven days:

```
SELECT LogDate,LogTime,UserName (FORMAT 'X(10)'),Event
FROM DBC.LogOnOff
WHERE Event NOT LIKE ('%Logo%') AND LogDate GT DATE - 7

ORDER BY LogDate, LogTime ;

LogDate   LogTime  UserName  Event
---------------------------------------------------------
07/07/30  08:55:22  S01a      BAD USER
07/10/21  08:59:53  BRM       BAD ACCOUNT
07/10/21  11:34:16  RPK       BAD PASSWORD
```

DBC.LogonRules View

Security administrators who have EXECUTE privilege on the DBC.LogonRule macro must specifically authorize a user to logon without a password. This is done with the SQL GRANT LOGON... WITH NULL PASSWORD statement.

The result of each successfully processed GRANT LOGON statement is stored as a row in DBC.LogonRuleTbl.

The DBC.LogonRules view returns the current logon rules.

For example, the following query requests all logon rule entries, sorted by username:

```
SELECT *
FROM DBC.LogonRules
ORDER BY UserName ;
```

The response shows that user SQL19 cannot log on using host ID 207, and that users SQL18 and SQL20 can log on without a password.

```
User Name         LogicalHostId       LogonStatus      NullPassword
--------------    ---------------     ------------     -----------
SQL18             1024                 G                T
SQL19             207                  R                F
SQL20             207                  G                T
```

DBC.Software_Event_Log

This view contains system error messages for Teradata Database Field Engineers. Rows are inserted by the system in response to software error conditions.

The messages also can contain information about Teradata Database feature software. For example, execution of the Performance Monitor commands SET RESOURCE, SET SESSION, and ABORT SESSION are considered major system events and thus are logged to DBC.Software_Event_Log. However, you may find more information in streams and event logs. (For more on the software event log view, see "Viewing the Software_Event_Log" on page 474.)

Tracking Access Using Query Bands

A query band is a set of name-value pairs assigned to a session or transaction 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 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.

When you enable query logging, query band information is stored in the DBC.DBQLogTbl table. Use the DBC.QryLog view to access this table.

```
BTEQ -- Enter your DBC/SQL request or BTEQ command:
sel queryband(FORMAT 'X(50)'), querytext(FORMAT 'X(30)') from qrylogv order by
 StartTime, queryid;

sel queryband(FORMAT 'X(50)'), querytext(FORMAT 'X(30)') from qrylogv order by
 StartTime, queryid;

 *** 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 QUERY_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 Chapter 13: “Tracking Processing Behavior with the Database Query Log (DBQL).”

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;

SELECT t1.sessionno, MonitorQueryBand(Hostid, sessionNo, runvprocno)
FROM TABLE (MonitorSession(1,'*',0)) AS t1;

 *** 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 query bands and how to assign them to a session or transaction, see SET QUERY_BAND in *SQL Reference: Data Definition Statements*. For more information on the MonitorQueryBand function, see *Workload Management API: Open and PM/APIs*.

Logging Access Attempts

Access Logging can check SQL requests submitted by users who attempt to access or execute data objects. Information can be tracked by:

- Type of access
- Type of request
- Requesting user
- Referenced object
- Frequency of access or attempted access

Note: Access checking and logging is very resource-intensive. Initialize this feature only if necessary and define rules on an as-needed basis.

You can only use access logging if you have already run the DIPACC script from the DIP Utility and it has created the DBC.AccLogRule security macro.

Enabling Access Logging

To enable logging of access checks, the DBC.AccLogRule macro must exist and you must have EXECUTE privilege on it. (See “Access Logging” in *Security Administration*.)

You activate access checking by defining the rules in one or more BEGIN LOGGING statements. The rules you specify are stored in system table DBC.AccLogRuleTbl. You can review all the rules currently in force by querying the DBC.AccLogRules view.

Every time a user defined for logging attempts to access an object, Teradata Database generates at least one entry in system table DBC.AccLogTbl. To review the access activity of current users, query the DBC.AccessLog view.

Disabling Access Logging

To disable access checking, use the following procedure:

1 Query the DBC.AccLogRules view to display the list of BEGIN LOGGING statements currently in effect:

```
select * from dbc.acclogrulesv;
```

2 Submit an END LOGGING statement for each BEGIN LOGGING statement in the list.

Note: To stop log activity completely, you need to issue one END LOGGING statement for every rule displayed (and then complete this procedure).

3 Issue a DROP MACRO for DBC.AccLogRule macro.

4 Do a TPA reset to set the change.

Caution: If you do not issue END LOGGING statements for every rule currently shown by the DBC.AccLogRules view, access logging continues even after you drop the DBC.AccLogRule macro and reset the database.

Viewing System Logging Information

Use the following views to find information about the entries in DBC.AccLogRuleTbl and DBC.AccLogTbl.

This system view ...	Contains ...	Based on the underlying table...
DBC.AccLogRules	current rules generated by BEGIN LOGGING statements. These entries determine what access checks should be performed.	DBC.AcctLogRuleTbl.
DBC.AccessLog	log entries (if DBC.AccLogRule macro exists) generated as a result of applying the rules. Each entry indicates a privilege check performed against a Teradata SQL request, based on the criteria defined by a BEGIN LOGGING statement.	DBC.AccLogTbl. **Note:** To control space consumption, empty this table regularly using the DBC.DeleteAccessLog view.

You can also query the Access Log records and include a join to the LogOnOff view. The following query would display SQL statements executed by user DBADMIN along with the IP address and OS user account for the logon that created the associated session:

```
SELECT a.UserName, a.LogDate, a.LogTime, StatementText, LogonSource
FROM DBC.AccessLog AS a, DBC.LogOnOff AS b
WHERE a.UserName = 'DBADMIN'
AND a.SessionNo = b.SessionNo
ORDER BY 2, 3
;
```

Restricting Logons Based on IP Address

Restrict logons of users based on IP address by creating a list of restricted IP addresses for the Gateway. The Gateway gets this list from a GDO and the GDO comes from one of two possible sources:

- For sites using a directory, the source is the directory itself.
- For sites that do not wish to use a directory, the source is an XML file.

Use the following utilities included with TDGSS to help create the GDO:

- The ipxml2bin utility reads the XML input defining the IP restrictions from the file and creates another file containing a "compiled" or more efficiently navigated binary data as a GDO.
- The ipdir2bin utility converts IP address restrictions contained in a directory to a GDO.

The number of IP restrictions allowed depends on the size constraint of the GDO. The GDO size is 128 KB but is affected by factors such as the length and number of names and IP addresses, number of actual filters, and the size of each filter.

Note: Logon attempts through any middleware application would cause the Gateway to see the IP address of the middleware application rather than the IP address of the true client. In this case, restrictions enforced by the Gateway would only be able to restrict the user based on the fact that the connection is coming in through the middleware application.

If an IP restriction is added or altered so that the combination of IP and user for a currently logged on session would be rejected if it were to newly log in, the session remains connected. If the database restarts, upon reconnect, previously logged on sessions still remain logged on. If, however, the user then logs off the session and tries to log back in, then the restriction will prevent the user from logging on.

While you can restrict users authenticated on the database and external users authenticated on a directory, you *cannot* restrict a profile or role based on IP address.

For more information and the syntax for the ipxml2bin and ipdir2bin utilities, see *Security Administration*.

Encryption

Logon and network data encryption are both available for the security of your data and system. For details on what you must set in the security mechanisms for enabling encryption, see "Managing Network Security" in *Security Administration*.

Logon Encryption

Logon encryption, which encrypts the entire logon string during logon processing, is automatically and transparently available.

The Gateway always supports SHA-256 encryption and uses SHA-256 encryption by default. For more information, see *Security Administration*

Password encryption of the logon string is supported on client platforms using ODBC, JDBC type-4, and CLIv2.

Network Data Encryption

Network data encryption is another way to control security on your system. All client protocols such as CLI, ODBC, JDBC, and OLE-DB, support network data encryption. However, check with the client application and utilities documentation to verify that they also support network data encryption.

For more information on how to enable network data encryption, see “Network Traffic Encryption” in *Security Administration*. For more information on CLIv2 and DBCAREA, see *Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems*.

Updating the Global Security Defaults

Control global security defaults by setting the fields of the DBC.SysSecDefaults table. Typically, you use the DBC.SecurityDefaults view to accomplish this. However, you must have the appropriate privileges to update the DBC.SecurityDefaults and the DBC.SysSecDefaults table directly. (If you have activated and set up a security administrator, you can log on as your special SecAdmin user; see *Security Administration* for instructions.)

Note: Changes to DBC.SysSecDefaults table settings do not take effect until after the system restarts. Note also that profile password settings take precedence over global settings.

To define your preferences, submit an UPDATE... SET statement against the table or view. For example, to set the maximum number of allowable logon attempts to 4 and an indefinite lockout for any user who exceeds that limit, you can submit:

```
UPDATE DBC.SecurityDefaults SET MaxlogonAttempts = 4;
UPDATE DBC.SecurityDefaults SET LockedUserExpire = -1;
```

Caution: The value in each SysSecDefaults column applies to all users, or, if your site implements profiles, to all users of any profile with that attribute set to NULL or NONE. If you set MaxLogonAttempts and LockedUserExpire at the system level, user DBC could potentially be locked out; yet only DBC can submit MODIFY USER DBC to change the DBC password. If this happens, call Teradata Support Center and have them unlock DBC through the TSTSQL console.

You can set up additional password options globally. For example, change the value in PasswordDigits field to require passwords to contain at least one numeric character or change the PasswordSpecChar field for special characters. You can also control if passwords must contain at least one alpha character, prohibit passwords from containing the username in the password string, and/or if user must define passwords with mixed case. See the DBC.SysSecDefaults table in *Data Dictionary* for the names of the fields you can set.

FOR information on...	SEE ...
the valid values for the DBC.SecurityDefaults fields and how to set them	*Security Administration*.
how to create a Security Administrator user	*Security Administration*.
how to set up and maintain a secure database environment	• *Database Design*. • *Security Administration*.
the system views associated with security and access control	• "Views That Report Explicit Privileges" on page 160. • "Tracking Access With Session-Related Views" on page 200. • *Data Dictionary*.
using roles and profiles to administer groups of users	• "Creating Roles" on page 135. • *Database Design*.

For More Information

The following table lists sources for more information.

IF you want more information on ...	THEN see ...
authorizing users defined on the database, that is, *not* defined on a directory server	"Using Roles to Manage Privileges" on page 134
external authentication	• "Single Sign-On" on page 188 • *Security Administration*
the role of the security administrator and how to implement a SecAdmin user	*Security Administration*
using REVOKE LOGON and GRANT LOGON statements	• "Logging Access Attempts" on page 206 • *Security Administration* • "SQL Data Control Language Statements" in *SQL Reference: Data Definition Statements*
using BEGIN LOGGING and END LOGGING statements	
running the DBS Control, Gateway Control, or DIP utility	*Utilities*

SECTION 3 Availability

CHAPTER 8 Protecting Data

This chapter discusses how to take advantage of the wide variety of automatic and optional data protection features provided by Teradata Database. This includes hardware and software data protection.

Automatic Data Protection Mechanisms

The Teradata Database system offers a variety of methods to protect data. Some data protection methods require that you set options when you create tables such as specifying fallback. Other methods are automatically activated when particular events occur in the system. Each data protection technique offers different types of advantages under different circumstances. The following list describes a few of automatic data protection methods:

- The Transient Journal (TJ) automatically protects data by storing the image of an existing row before a change is made, or the ID of a new row after an insert is made. It enables the snapshot to be copied back to, or a new row to be deleted from, the data table if a transaction fails or is aborted. (For details, see "Transient Journal (TJ) and DBC Space" on page 230.)
- Fallback protection is an optional data protection feature that you can specify so that the system automatically creates a copy of each row on another AMP is the same cluster. For more details, see "AMP Clustering and Fallback" on page 231.
- The Down AMP Recovery Journal supports fallback protection and automatically recovers data when an AMP is out of service or fails.
- Locks on data are automatically acquired during processing of a request. The four types of locks available are: exclusive, write, read, and access.

Transient Journal (TJ)

The TJ protects against failures that may occur during transaction processing. To safeguard the integrity of your data, the TJ stores:

- A snapshot of a row before an UPDATE or DELETE
- The row ID after an INSERT
- A control record for each CREATE and DROP statement
- Control records for certain operations

If a transaction is aborted or fails, the TJ enables the database to be restored to the state it was in before the transaction began. Its contents are used to:

- Copy snapshot rows back into their tables

- Remove inserted rows
- Delete partially created objects, or rebuild and, if necessary, repopulate dropped objects

FastLoad and MultiLoad jobs do not write to the TJ for individual rows and do not use the TJ to keep track of changes.

For more information, see “Transient Journal” on page 230.

Fallback Protection

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, the Teradata 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. See “AMP Clustering and Fallback” on page 231.

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

The benefits are summarized as follows:

- Permits access to table data when an AMP is offline.
- Adds a level of data protection beyond disk array RAID.
- Automatically applies changes to the offline AMP when it is back online.

For details on how recovery with fallback is accomplished, see “Down AMP Recovery Journal” on page 215.

Clustering AMPs

Clustering provides data protection at the system level. A cluster is a logical group of AMPs that provide fallback capability. If an AMP fails, the remaining AMPs in the same cluster do their own work plus the work of the down AMP. Teradata recommends the cluster size of 2.

Although AMPs are virtual processes and cannot experience a hardware failure, they can be “down” if the AMP cannot get to the data on the disk array. If two disks in a rank go down, an AMP will be unable to access its data, which is the only situation where an AMP will stay down.

A software problem can cause an AMP to go down and cause the database to restart. But as long as the AMP can access its disk, it should come back up during the restart. (For details, see “Restarts of the Teradata Database” on page 285.)

Warning: Two down AMPs in the same cluster cause the Teradata Database to halt. This requires a manual restart of the system.

For more details, see “AMP Clustering and Fallback” on page 231.

Down AMP Recovery Journal

The DownAMP Recovery Journal provides automatic data recovery on fallback-protected data tables when a clustered AMP is out of service. This journal consists of two system files stored in user DBC: DBC.ChangedRowJournal and DBC.OrdSysChngTable.

When a clustered AMP is out of service, the Down AMP Recovery Journal automatically captures changes to fallback-protected tables from the other AMPs in the cluster (see "AMP Clustering and Fallback" on page 231).

Each time a change is made to a fallback-protected row that has a copy that resides on a down AMP, the Down AMP Recovery Journal stores the table ID and row ID of the committed changes. When the AMP comes back online, Teradata Database opens the Down AMP Recovery Journal to update, or roll forward, any changes made while the AMP was down.

The recovery operation uses fallback rows to replace primary rows and primary rows to replace fallback rows. The journal ensures that the information on the fallback AMP and on the primary AMP is identical. Once the transfer of information is complete and verified, the Down AMP Recovery Journal is discarded automatically.

Note: Space for the Down AMP Recovery Journal is allocated from user DBC.

An AMP is placed out of service if two physical disks fail in a single rank. The AMP remains out of service until you replace the disks and run the Table Rebuild utility to reconstruct the table headers of fallback tables on the failed AMP. (See "Table Rebuild (rebuild)" in *Utilities*.)

Managing Locks

Locks are another form of data protection provided by Teradata Database. The following section gives an overview on locks. For extensive discussion on transaction locking, see "Locking and Transaction Processing" *SQL Reference: Statement and Transaction Processing*.

Locking Severities

Locking protects data from multiple users who are trying to make changes at the same time which keeps data integrity intact. This concurrency control is implemented automatically by the system. There are five types of locks as described in the following table.

Lock Severity	Description
Access	Prevents exclusive locks only. Caused by the LOCKING…FOR ACCESS statement modifier. This type of lock allows for reading data while modifications are in process. Access locks are designed for decision support on large tables that are updated only by small single-row changes. Access locks are sometimes called stale reads because you may get stale data that has not been updated.
Read	Prevents write and exclusive locks. Generally caused by a SELECT. Read locks ensure consistency during read operations. Several users may hold concurrent read locks on the same data, during which no modification of the data is permitted.

Lock Severity	Description
Write	Prevents other read, write, and exclusive locks. Generally caused by an INSERT, UPDATE, or DELETE statement. Write locks enable users to modify data while locking out all other users except those using access locks. (Users not concerned with data consistency or dirty reads can override the Write lock by using the LOCKING FOR ACCESS statement modifier.) 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 cause structural changes such as creating or dropping a column. Exclusive locks are the most restrictive type of lock. When you hold an exclusive lock, all other users cannot read or write to the data. Exclusive locks are most often used when structural changes are being made to the database. To create an exclusive lock, use the LOCKING modifier. When Archive/Recovery commands use exclusive locks, they are known as HUT locks.
Checksum	Placed in response to a user-defined LOCK FOR CHECKSUM when using cursors in embedded SQL and stored procedures. CHECKSUM locking is identical to ACCESS locking except that it adds checksums to the rows of a spool file to allow a test of whether a row in the cursor has been modified by another user or session at the time an update is being made through the cursor.

The Archive/Recovery (ARC) utility uses HUT[1] Exclusive locks when restoring, copying, rolling back, rolling forward, or building a database. Unlike with database locks, you must explicitly release HUT locks with the RELEASE LOCK option in the ARC utility command or by issuing a separate RELEASE LOCK in a script that sets HUT locks.

Client utilities (BTEQ, FastExport, FastLoad, MultiLoad, Teradata Parallel Data Pump, and Teradata Parallel Transporter) use standard database locks. For more information, see “Client Utility Locks and Teradata Database” in *SQL Reference: Statement and Transaction Processing*.

When using the RcvManager utility to cancel a transaction rollback, use the LOCKING FOR READ OVERRIDE modifier for the LOCKING request if you want to inspect the rows. For more information, see *SQL Reference: Data Manipulation Statements*.

Note: In some cases, you can override locks by using locking modifiers. However, these modifiers should *not* be used for OLTP because OLTP uses table level locks that incur significant overhead.

Lock Concurrency

The following table shows whether a new lock request is granted or must wait behind other locks that are either in queue or already in effect. For example, a new Read request must wait until the Write lock ahead of it is released before the new Read request can go into effect.

[1] For information on HUT locks, see *Teradata Archive/Recovery Utility Reference*.

	Level of Lock Held				
New Lock Request	None	Access HUT Access Checksum	Read HUT Read HUT Group Read	Write	Exclusive HUT Exclusive
Access HUT Access Checksum	Lock Granted	Lock Granted	Lock Granted	Lock Granted	Request Queued
Read HUT Read HUT Group Read	Lock Granted	Lock Granted	Lock Granted	Request Queued	Request Queued
Write	Lock Granted	Lock Granted	Request Queued	Request Queued	Request Queued
Exclusive HUT Exclusive	Lock Granted	Request Queued	Request Queued	Request Queued	Request Queued

An operation can wait for a requested lock indefinitely unless you specify the NOWAIT option. If you are working interactively and do not want to wait for a lock, use the .ABORT command in BTEQ to cancel the transaction.

Lock Levels

The system also can lock objects at a specific level when you use the LOCKING request modifier. The level of locking determines whether or not other users can access the same object. The following table lists the locking levels from the most to the least restrictive.

Lock Level	Description
Database	Applies to all tables and views in the database. All rows of all tables and their associated secondary index subtables are locked. This is the most restrictive level of locking.
Table/View	Applies to all rows in the table/view. All rows in a base table, all rows in any secondary index, and all rows in any fallback tables associated with the table are locked.
Row hash	The primary copy of rows sharing the same row hash value are locked. This is the least restrictive level of locking. A row hash lock permits other users to access other data in the table and a row hash is applied to all the rows that hash to a specific value (not a single row). **Note:** It is not necessary to lock the fallback copy of a row, nor any associated row, of a NUSI subtable.

The type and level of locks are automatically chosen by the system based on the type of SQL statement you issue. You can, however, upgrade a lock or downgrade a read lock to an access lock as necessary. Override locks by specifying the LOCKING request modifier in your SQL statements. For more information, see *SQL Reference: Data Manipulation Statements*.

If “dirty reads” are acceptable, use the locking severity of ACCESS at the locking level ROW hash as the best method to avoid deadlocks and minimize overhead.

Controlling Concurrency Through Isolation Levels

Isolation levels determine how a program or SQL statement acquires locks. To control the level for an individual request, use the LOCKING statement modifier. To control the isolation level for a session, use the TRANSACTION ISOLATION LEVEL statement.

You can change the default session-level lock by specifying:

```
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL
isolation_level_value;
```

where *isolation_level_value* can be either SERIALIZABLE (equivalent to READ lock) or READ UNCOMITTED (equivalent to ACCESS lock).

Use the following isolation level...	If you need...
READ UNCOMITTED	Fast response time over data accuracy. Response time is quick because queries can retrieve data without taking locks. Use READ UNCOMMITTED if you require read-only operations and it is not important that the data has not yet been committed. ("Read-only" operations do not include any SELECT statements specified within DELETE, INSERT, or UPDATE). READ UNCOMMITTED is especially useful for: • Looking up data that is generally static in nature, for example, codes or references in a look-up table. • Gathering information for statistical processing on a large amount of data when you only want to obtain averages or estimates for a general impression. **Note:** Using READ UNCOMITTED, the queries of the transaction will be reading data equivalent to ACCESS which may result in “dirty reads.” Dirty reads mean the data might be inaccurate if the data was in the process of being updated by some other transaction. Updates, inserts, and deletes are not affected because write operations obtain exclusive locks which are not released until the end of the transaction.
SERIALIZABLE	Accurate and committed data. SERIALIZABLE isolation provides a stable view of the database for SELECT transactions. For transactions containing UPDATE, INSERT, and DELETE queries, the SERIALIZABLE option causes the system to execute all transactions as though they were run one after another, even if they are run concurrently. Therefore, when processing locks, the system must wait for transactions to commit and may take longer to return results.

Note: You cannot use SET SESSION statements within a store procedure.

For more information, see “SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL” in *SQL Reference: Data Definition Statements*.

Hardware Data Protection

This section discusses the following topics:

- RAID technology
- Cliques
- Hot Standby Nodes

These features help protect against hardware failures.

Disk Arrays and RAID Technology

Redundant Array of Independent Disks (RAID) is a data protection scheme that uses additional (redundant) disks to protect data from a single disk failure.

An array is made up of the following:

- A set of disk drives attached to a disk array controller, either internally housed (mounted inside the system cabinet) or mounted in a separate cabinet
- Special array software (RAID Manager) that keeps track of how data is distributed across the drives
- A Disk Array Controller (DAC) cabled to a host adapter
- A Redundant Disk Array Controller (RDAC) cabled to a separate host adapter

Each array controller supports several channels. On an external array, each channel can have several disks. If the DAC or its host adapter fails, the RDAC can take over using a separate host adapter.

Cliques

A clique is a collection of nodes with shared access to the same disk arrays. Each multi-node system has at least one clique.

Nodes are interconnected via the BYNET. Nodes and disks are interconnected via shared buses and thus can communicate directly.

While the shared access is defined to the configuration, it is not actively used when the system is up and running. On a running system, each rank of disks is addressed by exactly one node.

The shared access allows the system to continue operating during a node failure. The vprocs remain operational and can access stored data.

If a node fails and then resets:

1. Teradata Database restarts across all the nodes.
2. Teradata Database recovers, the BYNET redistributes the vprocs of the node to the other nodes within the clique.
3. Processing continues while the node is being repaired.

Reducing Impact of Restarts with Hot Standby Node

You can greatly improve performance continuity by allowing the database to automatically switch between standby nodes and failed nodes of a production system. During the restart period caused by the failed node, vprocs can migrate from a failed node to the available nodes in the clique, including the newly joined hot standby node. When the failed node is repaired or recovered it becomes the new hot standby node.

Use the PUT utility to designate a node within a clique as the "hot spare." Once a node is assigned as the hot standby node, PUT will not assign any vprocs to that node. Assigning a hot standby node:

- helps with planned or unplanned restarts.
- eliminates the need for restarts to bring a failed node back into service. This eliminates the downtime and transaction aborts associated with a restart.

With a hot standby node as part of the configuration, the system always has a node available since it can use the spare node in place of the dead node to maintain the same performance as before.

As your clique sizes grow, re-assign fallback clustering. For more information on assigning fallback clustering or how to configure a hot standby node into a clique, see *Parallel Upgrade Tool (PUT) for Microsoft Windows User Guide* or *Parallel Upgrade Tool (PUT) for UNIX MP-RAS and Linux User Guide*.

Software Data Protection

The following sections discuss software data protection, including:

- Transaction data protection
- 2-Phase Commit (2PC) Protocol
- Checking disk I/O integrity
- Transient Journal protection
- AMP clustering and fallback
- Crashdumps and fallback
- Permanent journal (PJ) of before-image and after-image primary data rows
- Referential Integrity

Transaction Data Protection

The Teradata Database system protects data integrity by managing transactions to maintain consistent data for all users.

A transaction is a single unit of work. It is made up of one or more SQL statements.

An application-initiated asynchronous abort causes full transaction rollback. This type of abort is generated through a Call Level Interface Version 2 (CLIv2) abort request or is performed by the TDP when the application terminates without proper session cleanup.

Transaction Modes

Teradata Database supports two transaction modes:

- ANSI mode
- Teradata mode

You can alter the transaction mode via the SET SESSION command in Basic Teradata Query (BTEQ). See *Basic Teradata Query Reference* for more information.

ANSI Mode

If the system detects a statement error, or a privilege or false constraint violation, and all statements in a request cannot complete successfully, in ANSI mode the system:

- Aborts only the request
- Backs out any changes to mode by the request
- Does not release locks held by the request
- Returns an error response to the user

Note: If the statement in error is a DDL statement, not releasing the locks makes the system vulnerable to single-user deadlock between the locks placed by the DDL statement and those required for a parser dictionary cache request for a subsequent request from the same user.

If a deadlock or DDL error occurs, the system cannot successfully complete all statements in a request, and aborts the entire transaction.

To terminate a transaction, an application can execute an ABORT/ROLLBACK statement or a COMMIT statement (see "In-Doubt Transactions" on page 226). A BEGIN or END TRANSACTION statement in ANSI mode generates an error.

For more details on ANSI mode, see *SQL Reference: Fundamentals*.

Teradata Mode

In Teradata mode, the Teradata Database also automatically terminates transactions to preserve data integrity. If the system detects a statement error, deadlock, or a privilege or table constraint violation, and all statements in a request cannot complete successfully, the system:

- Aborts the entire transaction
- Backs out any changes to the database up to that point
- Releases locks
- Discards partially accumulated results (spool files)
- Returns an error response to the user

In Teradata Mode, transactions can be nested. That is, a transaction can contain another transaction. In a group of nested transactions, all transactions must complete successfully. If an error occurs anywhere within nested transactions, the system rolls back all changes made to the database to the point at which the first transaction began.

The COMMIT statement generates an error. If the SQL flagger is enabled, BEGIN/END TRANSACTION is flagged as non-ANSI.

Explicit and Implicit Modes

When running Teradata mode, also known as BTET mode, you can submit requests explicitly or implicitly:

- An explicit transaction consists of one or more requests enclosed by BEGIN TRANSACTION/END TRANSACTION statements.
- An implicit transaction is typically a macro, a data manipulation statement that affects a number of table rows, or a multistatement request that is not part of an explicit transaction, and for which the system automatically supplies BEGIN/END TRANSACTION statements.

See *SQL Reference: Data Manipulation Statements* for detailed syntax on BEGIN TRANSACTION and END TRANSACTION statements, and on transaction semantics.

2-Phase Commit (2PC) Protocol

2PC protocol is used by multiple systems that do not share the same locking and recovery mechanism to perform update transactions. Such systems, especially with more than one database management system, use 2PC to safeguard data in case of a failure.

A transaction using 2PC does not commit or roll back changes on its own, but does so only at the direction of an external coordinator. 2PC guarantees that all Teradata Database update transactions either commit or rollback.

The 2PC protocol allows customers to develop Customer Information Control System (CICS) and Information Management System (IMS) applications that can update one or more Teradata Databases or non-Teradata Databases in a synchronized manner. The result is that all updates requested in a defined unit of work either succeed or fail.

Participant and Coordinator

A coordinator manages processing for 2PC. When an application issues a commit or abort, the coordinator ensures that all participants either commit or abort.

A participant is a database manager that performs some work on behalf of the transaction, and commits or aborts changes to the database. A participant also can be a coordinator of participants at a lower level. In this case, the coordinator/participant relays a vote request to its participants, and sends its vote to the coordinator only after determining the outcome of its participants.

Note: Teradata Database is always a participant; it cannot be a coordinator. Also, Teradata Database transaction management differs between 2PC sessions and non-2PC sessions.

For the rest of this section, assume that one application is connected to the Teradata Database as a participant, and one application is designated as the coordinator.

2PC Protocol Requirements

To participate in the 2PC protocol with the Teradata Database, an application or coordinator must be able to execute the following functions via the CLI:

- Request a vote from the Teradata Database
- Issue a termination request

If a session is in 2PC mode, each transaction is implicitly started when one of the following occurs:

- A valid Teradata SQL request is received for the session after correct termination of a previous transaction.
- A valid Teradata SQL request (a syntactically and semantically correct Data Manipulation Language (DML) statement) is received initially for the session.

  Note: DDL statements, COMMENT, SET SESSION COLLATION, the DATABASE statement, and the CHECKPOINT statement are not allowed in 2PC mode.

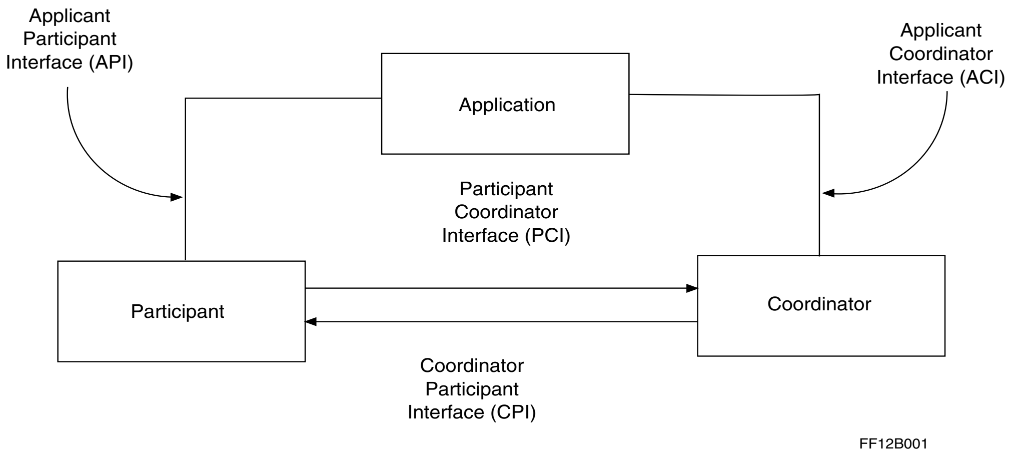
TDP and the Teradata Database provide functions to handle the resolution of in-doubt transactions following a Teradata Database reset or coordinator crash. These functions, which the coordinator initiates automatically or that you can initiate manually from a client terminal or the Teradata Database console, include the following:

- Produce a list of coordinators having in-doubt transactions.
- Produce a list of in-doubt transactions for a particular coordinator.
- Accept directives causing in-doubt transactions to be committed or aborted.

In 2PC mode, sessions may use only the syncpoint facilities of the coordinator. Sessions cannot use the BEGIN TRANSACTION, END TRANSACTION, and COMMIT WORK statements except in nested transactions.

2PC Protocol Components and Interfaces

The main components and interfaces involved with 2PC protocol appear in the following diagram.


Applicant Participant Interface (API)
Application
Applicant Coordinator Interface (ACI)
Participant Coordinator Interface (PCI)
Participant
Coordinator
Coordinator Participant Interface (CPI)
FF12B001

The functions of each interface appears as in the following table.

Interface	Description
ACI	Manages the communications between the application and coordinator. The application directs the coordinator to commit updates.
API	Used by the application for communication with the participant. The API performs tasks such as requesting 2PC sessions.
CPI	Can be considered the 2PC protocol. The CPI handles the vote requests and the abort and commit messages.
PCI	Manages the communications from the participant to the coordinator. These communications can include responses to requests for session information.

CICS and IMS act as coordinators, and in both systems the syncpoint manager coordinates the commit or rollback in all resources accessed in a logical unit of work.

Applications using CLIv2 may establish multiple 2PC sessions with one or more Teradata Database configurations.

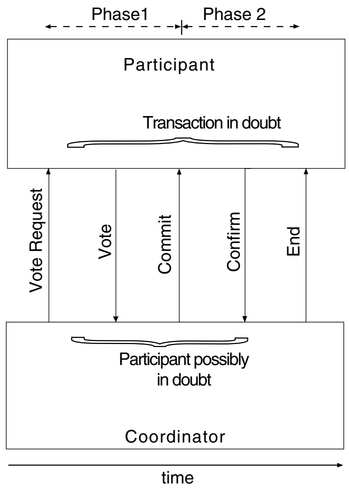
Applications written using the preprocessor can establish only one session at a time. Assembler, Common Business Oriented Language (COBOL), Programming Language/1 (PL/I), and C are supported for CICS and IMS applications.

An application requests to participate in the 2PC protocol at the time the Teradata Database session logs on. A CLI option can specify 2PC or non-2PC as the default startup mode.

Note: Multisession applications may use a mix of 2PC and non-2PC sessions. Sessions running in non-2PC mode are unaffected by 2PC operation.

Phase 1 is the voting phase of the protocol; phase 2 consists of committing or aborting the changes.

The processing scheme is illustrated in the following diagram:



FG11A003

FOR information on ...	SEE...
Phase 1 processing	following subsection.
Phase 2 processing	"2PC Processing - Phase 2" on page 226.
In-doubt transaction	"In-Doubt Transactions" on page 226 and "In-Doubt Resolution" on page 227.
In-doubt participant	

2PC Processing - Phase 1

At the start of each transaction, the initiating application generates a request for each prospective participant. Each participant is registered with the coordinator for the logical unit of work.

Upon receipt of the request, each participant determines if it can complete the transaction, then waits for instructions from the coordinator.

IF the ...	THEN ...
initiating application can make the update	**1** The application generates a commit request for the transaction. **2** Upon detection of the commit request, the coordinator sends, via the client interface, a vote request message to all participants. **3** The participants report whether they can or cannot commit the change to their individual databases.

IF the ...	THEN ...
transaction can be completed	the participant votes to commit. Once a participant votes to commit, it cannot change its vote. Therefore, before voting, each participant saves enough information in the TJ to enable it to subsequently commit or rollback the change, even if it crashes after voting to commit.
transaction cannot be completed	the participant votes to reject. For example, if the transaction of a participant failed before receiving the vote request, and the participant had already rolled the changes back, it would vote to reject.

2PC Processing - Phase 2

When the coordinator has received all votes, the transaction enters phase 2, in which the result of the vote is communicated to all participants.

The change can be committed only in phase 2. Based on the result of phase 1 voting, the coordinator sends a message to commit or abort to all participants.

IF ...	THEN the coordinator ...	AND each participant ...
all votes are to commit	logs any pending updates, then sends a commit message to all participants	1 makes the appropriate changes and releases any locks. 2 returns its status to the coordinator following the commit.
there is at least one vote to abort, or if the coordinator is unable to communicate with at least one of the participants	sends an abort message to all participants	1 aborts the operation and releases any locks held; no data is changed. 2 returns its status to the coordinator following the abort.

In-Doubt Transactions

A transaction is in doubt if any of the participants are in doubt. An in-doubt transaction remains inactive until an abort or commit is received from the host application.

Once a transaction is in doubt, the only valid request is to terminate (by aborting or committing) the transaction.

A participant is in doubt from the time ...	Until the time ...
the coordinator logs the vote of the participant	the coordinator logs the confirmation of the participant of transaction termination.
from the time it votes to commit	it receives and logs a response from the coordinator.

The coordinator considers the participant to be not in doubt once it has logged the confirmation of the participant of the completion of the unit of work. If a participant fails after phase 2 is initiated, that participant must perform the abort or commit processing after restart.

In-Doubt Resolution

The system performs automatic in-doubt resolution when communication is reestablished between the coordinator and participant. If the Teradata Database restarts, the system must re-establish in-doubt transactions as inactive transactions (note that non-read locks are held). At some point, the coordinator must send an abort or commit request to complete the transaction.

In some situations, you may need to manually resolve in-doubt transactions. You can perform manual in-doubt resolution via the:

- TPCCONS utility
- TDP COMMIT and ROLLBACK commands

2PC Error Handling

Errors are handled as described in the following table.

IF an error is detected ...	THEN the transaction is ...
before the transaction reaches the in-doubt stage	considered to be a failure and the transaction is aborted.
after the transaction reaches the in-doubt stage	not aborted and recovery may be possible.

Checking for Integrity of Disk I/O

Teradata Database can detect data corruption by sampling data from data blocks and generating checksums for individual tables or specific table types. If you enable checking for data integrity, Teradata Database generates and examines checksums on the data blocks.

A checksum is a computed value used to ensure that the data stored on a block of data is transmitted to and from storage without error. A sampling algorithm generates the checksum for a block of data which is then stored separately from the data on disk.

When the data is read back from disk, the Teradata Database file system recomputes the checksum based on the data read in and compares this value with the checksum that was previously calculated when the data was originally written out to disk. If the two values do not match, this indicates data corruption.

To enable disk I/O integrity checking on:

- An individual table, use the CHECKSUM option in the CREATE TABLE statement when creating a table. You can also use the ALTER TABLE statement to change the checksum level of an existing table. For more information, see *SQL Reference: Data Definition Statements*.
- Table type, use the DBS Control Utility. For more information, see *Utilities*.

Note: The system detects but does not fix data corruption. If the data is corrupted, contact your Teradata customer service representative to help restore the data.

Controlling the Checksum Generated

One method to control how much data is used to generate a checksum is on an individual table basis by specifying a checksum level with the CHECKSUM option in a CREATE TABLE or ALTER TABLE statement. The possible checksum levels to specify are as in the following table.

Disk I/O Integrity CHECKSUM Level	Default Checksum Sample Count	Description
DEFAULT	The checksum level defined in the checksum group of the DBS Control utility according to table type.	This is the default checksum level used when no CHECKSUM option is supplied in a CREATE TABLE, ALTER TABLE, CREATE HASH INDEX, and CREATE JOIN INDEX statement.
NONE	0%	Do not sample any data. This means the feature is disabled. Checksums are not generated and verified for this table.
LOW	2%	Sample a low percentage of each disk block to generate and verify a checksum.
MEDIUM	33%	Sample a medium percentage of each disk block to generate and verify a checksum.
HIGH	67%	Sample a high percentage of each disk block to generate and verify a checksum.
ALL	100%	Sample all of the data. This level samples all words in each disk block to generate and verify a checksum.

Change the checksum sample count percentage for LOW, MEDIUM, and HIGH by modifying the settings in the checksum group of the DBS Control utility.

Another method to control the amount of data used to generate a checksum is on a table type basis by modifying the table type checksum settings in the checksum group of the DBS Control utility. The following table types can be set to the NONE, LOW, MEDIUM, HIGH, and ALL checksum levels (the default checksum level for all table types is NONE):

- System Tables
- System Journal Tables
- System Logging Tables

- User Tables
- Permanent Journal Tables
- Temporary Tables

For more information, see "DBS Control Utility" in *Utilities*.

Impact on Performance Due to Checksums

As the number of words per disk used to generate and verify a checksum increases, the probability of detecting bit, byte, or byte string corruption increases. However, CPU utilization also increases and performance is impacted as more data is sampled.

But, note that even with the LOW checksum level (sample just one word per disk block by default), various forms of data corruption can still be detected. This includes all forms of whole disk sector corruption and *lost write* corruption. *Lost write* corruption occurs when data is written and the underlying storage reports back to the system that the write was successful, but the write actually never occurred. Set the checksum sampling to the level that best balances your performance and integrity needs.

Impact on Update In Place Operations Due to Checksums

When updating a table, the Teradata Database updates data *in place* if possible to improve performance. This means that modified data is written over the previous version of the data directly on disk. When updating in place, the write to disk must be done automatically to ensure data integrity. Update in place operations do not work with the disk I/O integrity checking because it is not possible to atomically update the data and the checksum for this data at the same time and still ensure data integrity. This is because the checksum is stored separately from the data, and is updated with a separate write operation. Therefore, when disk I/O integrity checking is enabled on a table, updates are not done *in place* and this can impact update performance.

When Modified Checksum Levels Take Effect

Changing checksum parameters does not have an immediate effect because the checksums and checksum levels are stored on disk. Changing the checksum level for an individual table or a table type only impacts future updates to existing tables and creation of new tables. For example, changing the user table type checksum level from NONE to LOW in the DBS Control utility does not cause all user tables to immediately start verifying checksums at the LOW level.

As the system updates rows in user tables and writes data to disk, checksums are generated for these updates using the newly selected checksum level. If you want to change the checksum level immediately on all tables of a certain table type, then use the UPDATE DATA INTEGRITY FOR command of the Ferret utility. This causes all tables of a certain table type to be read and rewritten with new checksums using the newly updated checksum level. If you want to change the checksum level on an individual table, then use the IMMEDIATE flag with the CHECKSUM option in the ALTER TABLE statement.

When Data Corruption is Detected

When the system detects corruption, the database restarts and does not allow access to the corrupt segment. If multiple accesses to a corrupt segment are attempted, fatal AMPs result. After the system detects a corruption in the data, contact your Teradata customer support representative who can:

- Determine if the corruption is due to a bad read, that is, if the data is correct on the disk.
- Compare the data with the data on a parity or mirrored disk. If the parity or mirrored data turns out to be okay, the data can be copied to the primary disk.
- Use the SCANDISK utility with the FIX option to fix the corruption if the corruption is minor. This works only if the file system metadata is corrupted and not the user data.
- Teradata Support Center personnel may use the Filer utility to repair the data if the corruption is in the user data but can be easily corrected.
- Restore the data from backup.

Transient Journal

Transient Journal (TJ) and DBC Space

The rows in TJ enable Teradata Database to roll back any changes made to the database by a transaction that is aborted or fails to meet a condition. The benefits of the TJ include the following:

- Always in effect
- Protects your data against transaction failures or aborts
- Inserts a new row each time a user submits a statement that changes the information in an existing table.
- Captures information about changes to the data rows and on the AMP.
- Automatically rolls back data to original values if you abort the transaction or there is a restart.
- Rollbacks on tables not using NUSIs have been optimized to block-at-a-time for user initiated aborts or system restarts.

The TJ maintains:

- Snapshots of rows in tables before a change is made (before image)
- Control records for dropped and created tables
- BEGIN and END TRANSACTION images

The TJ is always maintained by the AMPs as follows:

- Each AMP maintains its own TJ rows.
- Change information is under the control of the same AMP as the data row.
- An AMP background task periodically deletes TJ rows as soon as a transaction is either backed out or committed.

- If a DML statement is part of a multistatement transaction, the AMP does not delete the TJ blocks until one of the following is true:
  - Every statement making up that transaction is committed.
  - Rollback is complete for all processing performed when the transaction was active.

Note: To display the links between journal tables and the data tables that write to them, use the DBC.Journals view. The restricted version of this view displays only those tables that the requesting user owns or has privileges on.

Determining Maximum TJ Size

It is vitally important that DBC always has enough space to hold all the change rows generated by the total number of applications that run simultaneously during peak workload hours. As the administrator, you need to know that:

- The transient journal is maintained as a system table in database DBC.
- DBC PERM space is used to dynamically allocate space to the TJ.
- TJ data blocks are allocated as bytes of JournalDBSize. (Rows can be added to a TJ data block without a sector allocation. For details, see “DBS Control Utility” in *Utilities* and “JournalDBSize” in *Performance Management*).

If processing causes CurrentPerm of DBC to be exceeded, the transaction causing the overflow is not aborted due to lack of space. Thus, you have no way of knowing when TJ space is exhausted.

It is a good idea to determine how many rows the TJ needs to store during peak workload hours when the most jobs running simultaneously are changing data. TJ entries and the statements that generate them are as follows:

- Control records for CREATE and DROP
- Before-image rows for UPDATE and DELETE
- Row IDs for INSERT
- BEGIN/END TRANSACTION images

Note: Only UPDATE and DELETE cause a full row to be inserted into the TJ.

Use your estimate to determine whether DBC has enough CurrentPerm to support a maximum-sized TJ.

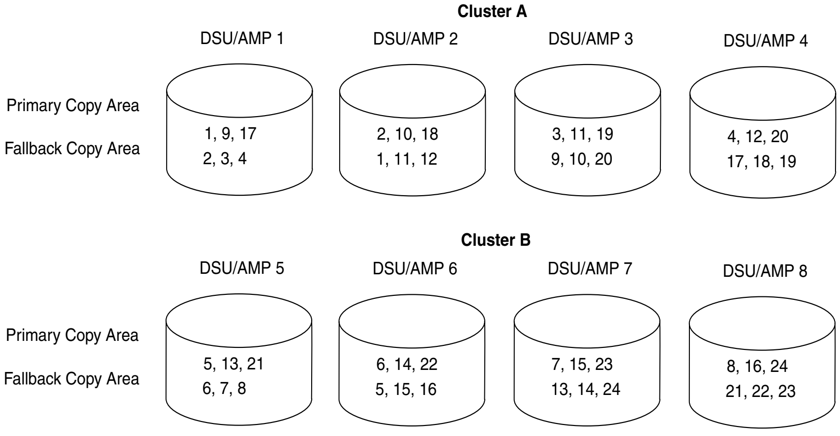
If it does not, the only short-term solution is to free up space. (For instructions, see “Permanent Space Availability” on page 90.) If you often need to do this, you might want to consider expansion.

AMP Clustering and Fallback

As described earlier in the chapter, fallback protection is an optional data protection feature accomplished by grouping AMPs into clusters. Within a cluster, a fallback copy of each data row is distributed to a different AMP from the one containing the primary row.

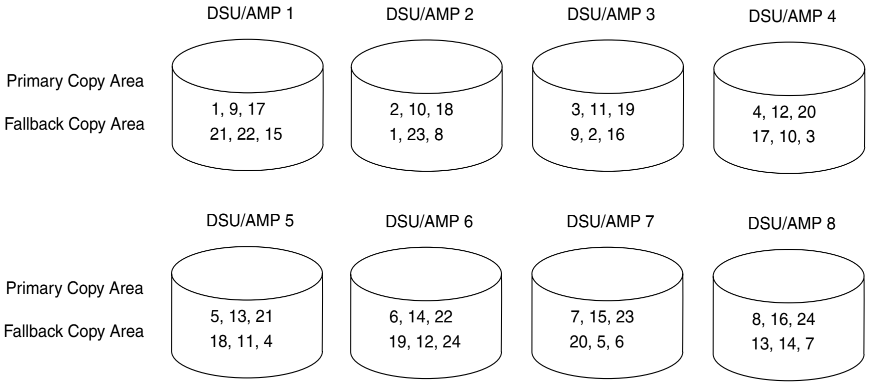
If the primary AMP fails, the system can still access data on the fallback AMP. This ensures that one copy of a row is available if one or more hardware or software failures occur within one rank, an entire array, or an entire node.

The following figure illustrates eight AMPs grouped into two clusters of four AMPs each. In this configuration, if AMP 3 (or its vdisk) fails and stays offline, its data remains available on AMPs 1, 2, and 4. Even if AMPs 3 and 5 fail simultaneously and remain offline, the data for each remains available on the other AMPs in its cluster.



FF06A005

The following figure illustrates a single eight-AMP cluster configuration, where primary and fallback copies of table rows are distributed across all eight AMPs:



FF06A004

The fallback copy of a row never resides on the same AMP as the primary copy. For example, if AMP 3 fails, its data remains available on AMPs 4, 5, and 6. However, if AMP 3 and AMP 5 both fail at the same time, row 11 is unavailable. This is why Teradata recommends you set up 2 clusters.

Note: For 7x24 systems, the fallback option on important tables is recommended for minimizing the risks of system downtime.

The following SQL statements demonstrate how to define fallback protection as the default for tables that a user creates.

The fallback option is actually activated when you use the CREATE TABLE or ALTER TABLE statements:

```
CREATE USER maxim
,AS PERMANENT =1000000,
,PASSWORD = mxm,
,FALLBACK;
```

or,

```
MODIFY USER maxim as FALLBACK ;
```

When a table is protected with the FALLBACK option, a copy of each permanent data row is stored on a separate AMP in the same cluster.

For information, see *Performance Management*.

Failure Handling

If two physical disks in the same rank fail, the associated AMP fails.

When an AMP fails, the system reads all rows it needs from the disks in the remaining AMPs in the cluster. If it needs to find a primary row from the failed AMP, it reads the fallback copy of that row on a disk in another AMP. The system updates fallback copies of rows.

To repair the failed AMP, you must replace the failed physical disks, and reconstruct the data from fallback copies of rows on functional AMPs in the cluster.

You can use the utilities listed in the following table to reconstruct data.

Utility	Description
RCVManager	Allows you to monitor recovery processing and transaction rollbacks. You can also use this utility to cancel rollback on specific tables. **Note:** To cancel rollback, you must first list rollback tables before running canceling rollback from the list. You must also list and cancel rollback within the same rcvmanager session.
Table Rebuild	Reconstructs tables on an AMP from data on other AMPs in the cluster.
Vproc Manager	Allows you to display and modify vproc states, and initiate a Teradata Database restart.

For more information on these utilities, see *Utilities*.

Activating Fallback Protection

You can activate fallback protection via the CREATE/MODIFY DATABASE/ USER and CREATE/ALTER TABLE statements. Following is an example of a statement that activates fallback protection for a new user:

```
CREATE USER maxim
,AS PERMANENT = 1000000
,PASSWORD = mxm
,FALLBACK;
```

Thus, fallback is the default for all tables created by Maxim in his own space. Maxim can override this default at the table level with the NO FALLBACK option of the CREATE TABLE or ALTER TABLE statement.

You can modify an existing database or user to use fallback with a MODIFY DATABASE…AS FALLBACK statement. However, MODIFY statements only affect tables that will be created in the future. Use ALTER TABLE to change the protection of existing tables.

Down AMP Recovery Journal

The system uses the Down AMP Recovery Journal for fallback-protected rows when an AMP is out of service. An AMP is placed out of service if two physical disk fail in a single rank. The AMP remains out of service until you replace the disks and reconstruct the data via the Table Rebuild utility. See *Utilities* for more information on the Table Rebuild utility.

Storage space for Down AMP Recovery Journal rows comes from user DBC PERM allocation.

This recovery journal keeps a copy of the tableIDs and rowIDs of fallback-protected rows on which changes have been made, and whose alternate copy (primary or fallback) would be under the control of the down AMP.

The Down AMP Recovery Journal is maintained by the other AMPs in the cluster. When the down AMP is returned to service, the system copies the indicated rows to disk from the other AMP, and discards the journal.

Changing Cluster Assignments

You can use the Configuration utility to reassign clusters into larger groupings, depending on the size of your configuration and available disk space. The following table lists some recommendations.

IF you ...	THEN ...
have a single-node system	Teradata recommends you retain the default cluster assignment.
change cluster assignments without adding AMPS or disks	make certain ample disk space is available on all AMPs. A general formula for recommended space is: `(CURRENTPERM * 3/2) should be less that 80% of total MAXPERM` For guidelines, see "Adding Vprocs" on page 413. For detailed information on using the Configuration Utility, see *Utilities*.

Spanning Clusters Across Cliques

It is desirable, when possible, to assign AMPs from different cliques to a cluster. Having each AMP in a separate clique improves the likelihood that data will remain available in the event a failure affects an entire clique.

Although such a failure is far less likely in a large configuration than the failure of a single node, a clique failure could have a serious impact. This is illustrated by the example clusters of two possible configurations (A and B) in the following tables.

Clusters, Configuration A

AMP IDs in Cluster	1	2	3	4 ... 8
	---	---	---	--- ---
	1-0	1-1	1-2	1-3 ... 1-7
	2-0	2-1	2-2	2-3 ... 2-7
	3-0	3-1	3-2	3-3 ... 3-7
	4-0	4-1	4-2	4-3 ... 4-7

Clusters, Configuration B

AMP IDs in Cluster	1	2	3	4 ... 8
	---	---	---	--- ---
	1-0	1-4	2-0	2-4 ... 4-4
	1-1	1-5	2-1	2-5 ... 4-5
	1-2	1-6	2-2	2-6 ... 4-6
	1-3	1-7	2-3	2-7 ... 4-7

In this example, assume that each configuration has 4 cliques, each containing 8 AMPs, with 8 clusters of 4 AMPs each. Each AMP in each cluster is identified by a unique vprocid. Vprocids are shown in the format c-p (clique number-vprocid) to better illustrate the issue. The actual format is *nnnn*, where a vprocid is in the range 0-1023.

In configuration A, AMPs in each cluster are spread across cabinets; therefore, failure of a cabinet results in the loss of just a single AMP in each cluster. If there are no other processor failures, the system continues to run because primary data for AMPs in the failed cabinet remains available on the fallback AMPs in the other cabinets.

In configuration B, failure of a cabinet results in the loss of all AMPs in two clusters. This causes the entire system to fail because primary and fallback data is unavailable for two clusters.

Spanning Clusters Across Arrays

For multi-clique configurations, you can define clusters so that each AMP in a cluster comes from a different clique. This grouping of clusters protects all your data rows, even if an entire clique should fail. In this case, the cluster size can be no larger than the total number of cliques.

Archiving by Cluster

The Archive/Recovery (ARC) utility allows backup and restore of data from clustered AMPs. A CLUSTER option is available on both the DUMP and the RESTORE commands. For more information on ARC, see Chapter 9: "Archiving, Restoring, and Recovering Data" and *Teradata Archive/Recovery Utility Reference*.

Because non-local PJs enable data to be rolled forward after tables are restored, the JOURNAL option (in the CREATE USER/DATABASE/TABLE statement) is recommended if you decide to define clusters.

The following section explains the advantages of permanent journaling.

Crashdumps and Fallback Protection

DIP creates the Crashdumps database with the FALLBACK protection option. Thus, because tables default to the same protection defined for the database in which they reside, PDE dumps are saved as fallback tables.

Accessing Dump Data

The FALLBACK option enables dumps to be saved in the database when an AMP is down.

This means that even if the Teradata Database cannot come up with all vprocs, you can still access current dump data with the client-resident Dump Load/Unload (DUL) utility to find the cause of the problem. For more information on the DUL utility, see *Utilities*.

Dump Handling With No Fallback

To save space, you can modify Crashdumps temporarily to remove fallback.

Caution: Without fallback, you cannot access dump data if an AMP is down.

If an AMP goes down and Crashdumps has no fallback protection, the system handles a dump as follows:

- The system will not copy any raw PDE dump data from the dump device areas into Crashdumps or Windows flat files until all AMPs are online.
- If the copy operation is already in progress when an AMP goes down, the copy operation is deferred until after a restart brings the AMP online.

For more information about crashdumps, see Chapter 15: "Handling Teradata Crashdumps."

Permanent Journal (PJ) Protection

A Permanent Journal (PJ) 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. A database/user can contain only one PJ.

PJ rows are under your control, so you can checkpoint, archive, and restore them. PJs remain available until you delete them. PJ tables provide protection against:

- Loss of data caused by a disk failure in a table that is not fallback- or RAID-protected
- Loss of data if two or more AMPs fail in the same cluster (that is, loss of two disks in a rank/failed AMP)
- Incorrect operation of a batch or application program
- Loss of changes made after a data table is archived
- With dual journaling, loss of one copy of the journal table

PJ also allows disaster recovery of an entire system.

PJ Options

A PJ provides the options as listed in the following table.

Option	Description	Change Image Location	
		Fallback Tables	Non-Fallback Tables
Single before-image	• Captures image before a data row change • Provides protection against software failure • Allows rollback to a checkpoint	Primary AMP and fallback AMP	Primary AMP
Single after-image	• Captures image after a data row change • Provides protection against hardware failure • Allows rollforward to a checkpoint	Primary AMP and fallback AMP	Backup AMP
Dual before-image	• Maintains two copies of an image before a data row change • Provides protection against journal loss		Primary AMP and backup AMP
Dual after-image	• Maintains two copies of an image after a data row change • Provides protection against journal loss		Primary AMP and backup AMP

Journaling of Fallback Tables

If you specify PJ for a table with fallback, the journal rows are also fallback-protected. That is, regardless of the type of journaling selected, each journal row is always written to both the primary disk and the corresponding fallback disk.

Fallback Versus PJ

PJ is not a substitute for fallback or RAID. The difference can be summarized as follows:

- Fallback and RAID maintain a duplicate copy of the current image of every row in the data table.
- The PJ maintains before and after image rows that the system uses to roll table data forward or backward in time.

Space Allocation

A PJ requires permanent space. The system uses fallback hash maps to determine the fallback AMP for the journal row. A fallback AMP is another AMP in the same cluster.

A backup AMP is another AMP in the same cluster as the primary AMP. The system does not use a hashing algorithm for backup AMPs. All images for one AMP go to a single backup in the same cluster.

For more information on PJ options, see the CREATE TABLE and the ALTER TABLE statements in *SQL Reference: Data Definition Statements*.

PJs have the following effects on recovery from a single disk failure.

Protection	Notes
Fallback tables, dual image journals	Data is fully available. Journals play no part in recovery.
No fallback tables, dual image journals	Data is partially available. You may fully recover data and journals.
No fallback tables, single image journals	Data is partially available. You may recover data, but the journals are lost.
No fallback tables, no journals	Data is partially available. You can recover data only to the point of the last archive.

Deleting a PJ

You might want to delete a PJ if no table has been assigned to use that journal, or if you chose to use another PJ for administrative reasons.

However, be very cautious that no data tables are still defined to write to that journal when you delete it.

An archive of your database cannot be restored if it contains tables defined to write to a journal table that does not exist in the archive.

To delete a PJ:

1 Submit an ALTER TABLE statement to stop the journaling to that PJ:

```
ALTER TABLE table_name
WITH JOURNAL TABLE = journal_table_name
```

```
,NO BEFORE JOURNAL
,NO AFTER JOURNAL;
```

2 Submit a MODIFY DATABASE/MODIFY USER statement; for example:

```
MODIFY DATABASE database_name AS
DROP DEFAULT JOURNAL TABLE = journal_table_name;
```

Replicating Your Server

Using Teradata Replication Solutions - GoldenGate Replication Products, you can capture changes made to a specific set of tables on your production system and apply those changes to a target server. If the production system is unavailable for an extended period, you can continue critical application processing on replication servers that duplicate the production system.

GoldenGate™ for Teradata helps facilitate replication by serving as the intermediary for a primary Teradata Database server and a subscriber server, which may or may not be another Teradata Database server. Database updates are applied to the subscriber system only if the source transaction completes successfully. You can also create user exits for further data processing. For more information on GoldenGate for Teradata, see *GoldenGate Administrator Guide* and *Teradata Replication Solutions Overview*.

To replicate changes, define the group of tables you want to capture changes for using the CREATE REPLICATION GROUP statement. For more information on the SQL syntax for defining, changing, and using replication groups, see *SQL Reference: Data Definition Statements*.

Note: The intermediary software is not part of a Teradata release. To purchase and set up Teradata Replication Solutions, contact your customer representative.

Required Setup Tasks

Before you can use Teradata Replication Solutions, you must do the following:

- Ensure that sufficient disk space exists for the directory that contains Relay Services Gateway (RSG) spill files for every node in the system. The default path is /var/tdrsg on MP-RAS, /opt/teradata/tdat/temp/tdrsg for Linux, and <TDAT>\tdTemp\tdrsg for Windows where <TDAT> is typically C:\Program Files\NCR\TDAT. However, be sure to check if this path was set differently during system installation for your site.
- Define RSG vprocs. New RSG virtual processors (vprocs) help drive the server tasks for Replication Solutions. There must be one RSG vproc on each TPA node in the server. Use the Parallel Upgrade Tool (PUT) to add a RSG to every node. For more information, see *Parallel Upgrade Tool (PUT) for Windows* User Guide.

Considerations for Replication Groups

Tables defined in a Replication Group *cannot* contain the following:

- LOB defined on any column
- A column defined for generation of an Identity Column with GENERATED ALWAYS. However, defining a table with an identity column will not result in an error if they are defined with GENERATED BY DEFAULT.

You can define triggers on a replicated table. However, if both the source and subscriber tables define the same trigger, the triggered action might be applied twice to the subscriber table (once through replication, and a second time when the triggering action is applied via SQL on the subscriber). You must ensure that the triggers are designed with appropriate logic to prevent unwanted actions under replication.

For more information, see *GoldenGate Administrator Guide* and *Teradata Replication Solutions Overview*.

Using Referential Integrity

Define a referential constraint on a table with the REFERENCES clause of the CREATE/ ALTER TABLE statement.

Depending on your choice of the associated options, a REFERENCES clause can apply to three forms of referential constraints,[2] as in the following table.

Referential Constraint Type	DDL Clause	Level of Referential Integrity Enforcement	Application
Referential integrity constraint	REFERENCES	Row	• Tests each row during an insert, delete, or update operation. • If the constraint would be violated, the AMP rejects the operation and returns an error message.
Batch referential integrity constraint	REFERENCES WITH CHECK OPTION	Implicit transaction	• Tests each row throughout an entire transaction during an insert, update, or delete operation. • If the constraint would be violated by any row, the Parser aborts the operation, rolls back the transaction, and returns an abort message.
Referential constraint	REFERENCES WITH NO CHECK OPTION	None	• Does not test for referential integrity. • Assumes that the user enforces data integrity with some other method.

[2] When printed in lowercase letters, the term referential constraint refers to the general class of referential constraints, including standard referential integrity, batch referential integrity, and referential constraints. When printed in first-letter capitals, the term Referential Constraint refers specifically to a constraint that defines a referential relationship but instructs Teradata Database not to enforce it.

Each type is briefly introduced in the rest of this section, emphasizing aspects that might be of special interest to you as the DBA.

Note: Special optimization of certain queries is possible with each type. For more information, plus examples of use, see "REFERENCES" and "Referential Constraints" in CREATE TABLE in *SQL Reference: Data Definition Statements*.

Standard Referential Integrity Constraint

A standard referential integrity (RI) constraint enforces the rule that a non-null value in a referencing column (the foreign key) can exist only when an equal value exists in the column being referenced (the primary or alternate key).

The columns in a referencing (Child) table are the foreign-key (FK) columns to the candidate columns (primary key (PK) or alternate key) in the referenced (Parent) table. The columns in both tables should be defined as NOT NULL and must be identical in name, data type, quantity, and case sensitivity.

The following table lists RI constraints for each column-level REFERENCES clause in a CREATE TABLE or ALTER TABLE statement.

IF you ...	THEN the...
omit column_name	Parent table must have a single-column primary key; the FK column references that column by default.
specify column_name	name must reference: • The single-column primary key of the Parent table • A single-column alternate key in the Parent table, defined as UNIQUE

Standard referential integrity checks the equality row by row, ensuring data integrity during insert, delete, or update operations on a table. However, this fine granularity incurs a modest performance overhead.

More explanation of the key columns and how to choose them and the referential integrity rules and how to apply them are given in "FOREIGN KEY" and "REFERENCES" under CREATE TABLE and ALTER TABLE in *SQL Reference: Data Definition Statements.*

Note: Foreign key references are not allowed in Multiload or FastLoad target tables.

Batch Referential Integrity Constraint

A batch referential integrity constraint is less expensive to enforce than standard referential integrity because checking is performed on qualified rows within a transaction rather than on all rows in the table.

When the reference point is reached, the parser joins the Parent and Child rows and then tests them. If a violation is found, all statements in the transaction are rolled back.

Thus, the enhanced performance can incur the following costs:

- With very large tables, a rollback can be expensive. You should use Batch RI only for smaller tables, or those whose normal workloads will not trigger reference violations.
- Query results might be inaccurate, depending on the type and amount of operations in the transaction and how deeply into the transaction the first violation is detected.
- Utilities like FastLoad and MultiLoad are not able to operate on tables defined for batch referential integrity checking.

Referential Constraint Using WITH NO CHECK OPTION

No constraints are enforced when you use the WITH NO CHECK OPTION with the REFERENCES clause in the CREATE TABLE or ALTER TABLE statement. Referential Constraint allows referential integrity to be used to advantage by the Optimizer, but does not incur the overhead of the database-enforced referential integrity.

Caution: WITH NO CHECK OPTION does not enforce the referential constraints. It is possible, then, that incorrect results or corrupt data could occur, unless you take other measures to ensure that data integrity is maintained. You should allow NO CHECK only when the possibility of data corruption or deletion and erroneous query responses are not critical or can be prevented. For full details and examples, see "Validating the Integrity of Base Tables In a Referential Constraint Relationship" and "Scenario for Data Corruption With Referential Constraints" in *SQL Reference: Data Definition Statements*.

Use care when manipulating data within a NO CHECK environment. NO CHECK means that a row having a non-null value in a FK column is allowed to exist in a Child table when an equal value does not exist in the PK or alternate column of the Parent table. Also, DML operations are allowed on NO CHECK tables that cannot be performed on referential integrity tables.

For example, if a referential relationship is defined using NO CHECK, and an INSERT, DELETE, or UPDATE statement containing a redundant RI join is applied when the PK-FK row pairs for the eliminated join do not match, the operation is allowed.

Data in the Parent tables of these relationships can be deleted or corrupted. Depending on the operation, no warning is given if such an error occurs.

Operational Behavior

The following table lists operational behavior for RI with WITH NO CHECK OPTION.

IF you use NO CHECK tables with ...	THEN ...
FastLoad and MultiLoad	• Reference violations are not checked. • Flags are not set, warning are not issued, and error messages are not returned if violations exist.
INSERT, UPDATE, and DELETE processing	• Reference violations are not checked. • Flags are not set, warning are not issued, and error messages are not returned if violations exist.

IF you use NO CHECK tables with ...	THEN ...
ARC	there is no change in operation, behavior, or any RI-related error messages. You can use the REVALIDATE REFERENCES FOR command to reset the state of the table after a RESTORE operation. **Note:** The REVALIDATE REFERENCES statement does not validate "soft" RI tables (tables with NO CHECK option specified).
CheckTable	CheckTable ignores constraints on NO CHECK tables and sends a message that RI checks are being skipped. Otherwise, there is no change in operation or behavior.

Data Validation

If you plan to bypass references checking, you may prefer to use a procedural constraint, such as a set of triggers, to handle inserts, updates, and deletions. If you do not enforce some type of reliable integrity constraint, Teradata recommends you establish a validation procedure that includes:

- A query to test for and report referential integrity violations
- A transaction to correct the reported violations

The query should be run every time data is changed. A good validation procedure could be based on the following:

1 Run one DML operation, such as an update, on RI/NO CHECK tables.

2 Before performing another operation on those tables, run the validation query against each of them to find corrupt rows.

The following query structure reports every row in the Child table with a foreign key value that does not match a primary key value in the Parent table. (Foreign key nulls are specifically excluded because it is not possible to determine what values they represent.)

```
SELECT DISTINCT childtablename.*
FROM childtablename, parenttablename
WHERE childtablename.fkcol NOT IN
(SELECT pkcol FROM parenttablename)
AND childtablename.fkcol IS NOT NULL;
```

3 Delete from the Child table any reported rows as soon as possible in order to maintain the integrity of your database.

4 Run the validation query again immediately after performing any updates or deletes on RI tables using WITH NO CHECK OPTION.

For detailed examples and a thorough discussion of your responsibilities in maintaining data integrity when you use REFERENCES WITH NO CHECK OPTION, see the following:

- Under “CREATE TABLE (Column Definition Clause)” in *SQL Reference: Data Definition Statements*: “Validating the Integrity of Base Tables In a Referential Constraint Relationship” and “Scenario for Data Corruption With Referential Constraints”
- Declarative constraints versus procedural constraints (such as triggers), under "Designing for Database Integrity" in *Database Design*.

Table Header Referential Indexes (RIs)

When you define REFERENCES for a table, information about referential integrity is maintained as index fields in the table headers of both the Child and the Parent tables.

The RI in a Child table is made up of the fields listed in the following table.

<table>
<tr><th>Child RI Fields</th><th>Contents</th></tr>
<tr><td>IndexID</td><td>Reference index number.</td></tr>
<tr><td>IndexType</td><td>The AMP index type (AMPIT), which for this index is REFERENCE.</td></tr>
<tr><td>State</td><td>Describes the current state of the RI as one of the following:
<table>
<tr><th>This value ...</th><th>Indicates a state that is ...</th></tr>
<tr><td>validRI</td><td>normal.</td></tr>
<tr><td>invalidRI</td><td>the result of a rollforward or rollback.</td></tr>
<tr><td>inconsistentRI</td><td>the consequence of a RESTORE operation.
Use the ARC REVALIDATE REFERENCES FOR command to revalidate the index.
Note: The REVALIDATE REFRENCES statement does not validate "soft" (NO CHECK option) RI tables.</td></tr>
<tr><td>unresolvedRI</td><td>the result of a forward reference, where a Child table references a nonexistent Parent table.
Once the Parent table is created, the unresolvedRI value changes to validRI.</td></tr>
</table></td></tr>
<tr><td>DBID</td><td>Identifies the database in which the Parent table resides (the Parent owner database). The value depends on the state of this Child table:
<table>
<tr><th>IF the state of this table is ...</th><th>THEN the value of DBID is ...</th></tr>
<tr><td>valid, invalid, or inconsistent</td><td>ID of the database in which the Parent table resides.</td></tr>
<tr><td>unresolved</td><td>Name of the Parent table, until the Parent table is created.</td></tr>
</table></td></tr>
<tr><td>ChildEntry</td><td>The value TRUE.</td></tr>
</table>

<table>
<tr><th>Child RI Fields</th><th>Contents</th></tr>
<tr><td>TableID</td><td>Identifies the Parent table, based on the state of this Child table:
<table>
<tr><th>IF the state of the table is ...</th><th>THEN the value of TableID is ...</th></tr>
<tr><td>valid, invalid, or inconsistent</td><td>ID of the Parent table.</td></tr>
<tr><td>unresolved</td><td>AMPNullUsrTblID (null), until the Parent table is created. At that time, the state of this table becomes valid.</td></tr>
</table></td></tr>
<tr><td>SecIdxId</td><td>The SI number of the corresponding PK or alternate key column, based on the definition of the key columns and the state of this table:
<table>
<tr><th>IF the...</th><th>THEN the value of SecIdxID is ...</th></tr>
<tr><td>PK or alternate key column is not the PI, or is a nonunique PI (NUPI or NUPPI)</td><td>the SI number of the corresponding PK or alternate key column.</td></tr>
<tr><td>PK or alternate key column is defined as the PI and unique (a UPI or UPPI)</td><td>0 (zero).</td></tr>
<tr><td>table is in an unresolved state</td><td>0 (zero), until the Parent table is created.</td></tr>
</table></td></tr>
<tr><td>FKFields</td><td>Describes the FK columns.</td></tr>
</table>

The RI in a Parent table is made up of the fields in the following table.

<table>
<tr><th>Parent RI Fields</th><th>Contents</th></tr>
<tr><td>IndexID</td><td>Reference index number.</td></tr>
<tr><td>IndexType</td><td>The AMPIT, which for this index is REFERENCE.</td></tr>
<tr><td>State</td><td>Describes the current state of the referential index (RI) as one of:
<table>
<tr><th>This value ...</th><th>Indicates a state that is ...</th></tr>
<tr><td>ValidRI</td><td>normal.</td></tr>
<tr><td>InvalidRI</td><td>the consequence of a rollforward or rollback operation.</td></tr>
<tr><td>InconsistentRI</td><td>the consequence of a RESTORE operation. (Use the ARC REVALIDATE REFERENCES FOR command to revalidate the RI.)
<b>Note:</b> The REVALIDATE REFRENCES statement does not validate "soft" (NO CHECK option) RI tables.</td></tr>
</table></td></tr>
</table>

Parent RI Fields	Contents
DBID	The ID of the database in which the Child table resides (the Child owner database).
TableID	ID of the Child table.
ChildEntry	The value FALSE.
SecIdxId	Secondary index number, or 0 (zero) if the PK column(s) (or alternate key column) in this table are defined as a UPI or UPPI.
FKFields	Describes the FK columns.

Comparable state settings indicate the state of a Referential Constraint table:

- ValidSoftRI
- InvalidSoftRI
- UnresolvedSoftRI
- InconsistentSoftRI

Comparable state settings indicate the state of a Batch RI table:

- ValidBatchRI
- InvalidBatchRI
- InconsistentBatchRI
- UnResolvedBatchRI

The meaning and rules for SoftRI and BatchRI states are the same as for RI states.

Some operations depend on the state of the RI for success. For example, if the state is Invalid, Inconsistent, or Unresolved, the following operations are not allowed:

- Update, insert, or delete operations
- ARC ARCHIVE/RESTORE or COPY/RESTORE

Use the REVALIDATE REFERENCES FOR command to reset the state of a table that is set to "inconsistent" as a result of RESTORE operations. This, however, does not apply to tables using soft RI (NO CHECK option). REVALIDATE REFERENCES FOR statement does *not* validate RI tables using the NO CHECK option.

For more information on revalidating references, see "Changing the State of RI Tables with REVALIDATE REFERENCES FOR" on page 268.

Tracking RI Definitions

You can track the definitions of referential constraints with the views in the following table.

This view ...	Provides information about ...
DBC.All_RI_Children	defined referential constraints from the child-parent perspective.

This view ...	Provides information about ...
DBC.All_RI_Parents	defined referential constraints from the parent-child perspective.
DBC.RI_Child_Tables	tables in child-parent order. This view is similar to the All_RI_Children view, but returns the internal IDs of databases, tables, and columns.
DBC.RI_Distinct_Children	tables in child-parent order without the duplication that could result from multi-column FKs.
DBC.RI_Distinct_Parents	tables in parent-child order without the duplication that could result from multi-column FKs.
DBC.RI_Parent_Tables	tables in parent-child order. This view is similar to the All_RI_Parents view, but returns the internal IDs of databases, tables, and columns.

Restrictions

Referential integrity checking is *not* supported at all for use with:

- FastLoad
- MultiLoad
- COMPRESS values
- Identity columns
- Column-level constraints (other than reference) on PK and FK columns
- Global temporary and volatile tables

If such operations are attempted on a Child or Parent table, an error message is returned.

Note: Performing a restore or copy of selected partitions marks referential integrity references as inconsistent. You must use the REVALIDATE REFERENCES FOR to correct header indexes for referential integrity references that have been marked as inconsistent. the consequence of a RESTORE operation.

The REVALIDATE REFRENCES statement does not validate soft (NO CHECK option) RI tables.

Guidelines for Choosing Data Protection

When selecting which data protection to use, consider the guidelines in the following table.

IF ...	THEN ...
• you have a Teradata server than can act as the subscriber server • you need a system that virtually never fails • your system processes low-volume, random update activity	use replication solutions. Systems that use batch-style updates (using TPump or MultiLoad) are more efficiently done using a "dual loading" strategy. **Note:** The intermediary software is not part of a Teradata release. You can purchase the GoldenGate for Teradata software either from Teradata or directly from GoldenGate.
• you require high data availability • disk space is not a concern	use fallback.
• the table is not fallback-protected, and data maintenance is performed on the Teradata Database • you require the ability to back out committed changes to a specific time • large production tables must be archived while production continues	use PJ.

The following table summarizes the advantages and disadvantages of data protection alternatives.

Protection Mode	Advantages	Disadvantages
Fallback	• Transparent • Requires no manual intervention. • Data is fully available after a failure.	• Requires twice as much disk space. • Requires twice as many write I/O operations as only using PJ. • Requires more than twice as much processing time to update (insert, update, and delete) rows. CPU utilization may increase 20% to 40% using fallback.
No fallback	• Requires less disk space. • Requires fewer I/Os.	• Data not fully available after a failure. • Requires reloading if a failure occurs.
PJ	• No writes to the TJ are required for PJ images. • A single-image journal is as fast, or faster, than fallback without PJ. • Requires less disk space (if properly managed).	• Requires human intervention. • Dual image journaling is slower than fallback.

CHAPTER 9 Archiving, Restoring, and Recovering Data

This chapter discusses how to archive, restore, and recover data on Teradata Database specifically with the Archive/Recovery (ARC) utility. The ARC utility, through its associated script language, provides the link between Open Teradata Backup (OTB) solutions and Teradata Database.

To use ARC on a mainframe, you must provide the overall management of the backup, archive, and restore of files at the mainframe. Teradata does not have a role in determining what overall backup, archive, and recovery solution is used on the mainframe. On a mainframe, ARC provides the capability to move the database objects between the mainframe and Teradata Database.

In general, the direction with all OTB solutions is to provide automation in the generation of ARC scripts requiring less detailed knowledge of the ARC utility to perform the typical archive, restore, and recover tasks. Each OTB provides varying degrees of support towards this direction. You will need a detailed understanding of ARC when it is necessary to analyze problems in archive, restore, and recover operations or it is necessary to develop ARC scripts to perform operations that are not supported by the particular automated script generation of your OTB solution.

The topics discussed in this chapter include:

- The phases of archiving and restoring data
- Recommended ARC routines
- Archive-related statements
- Archiving Data Dictionary tables in the DBC database
- Restore-related statements
- Restoring after an AMP goes down
- Considerations when restoring tables with partitioning
- Using the CHECKPOINT statement
- Recovering data using ROLLFORWARD/ROLLBACKWARD operations
- Canceling rollback
- Using the views of the Recovery Control Catalog (RCC)

For a comparison of import and export utilities, including Archive/Recovery, see "Client Utilities Feature Comparison" in Appendix B: “Import/Export Utilities.”

Overview

Open Teradata Backup solutions provide the interface to invoke the Archive/Recovery (ARC) utility. Through the ARC script language, OTB solutions can archive, restore, recover, and copy data. ARC for UNIX MP-RAS users or the combination of the OTB and ARC for Windows users is used to:

- Archive a database, individual database table, or selected partitions from Teradata Database to tape.
- Restore a database, individual database table, or selected partitions to Teradata Database from tape.
- Restore a copied database, table, or selected partitions to a different Teradata Database.
- Recover a database to an arbitrary checkpoint by rolling it backward or forward using before- or after-change images in a permanent journal table.
- Delete a changed-image row from a permanent journal table.

When to Use ARC

Teradata Database provides a number of automatic data protection features. However, these features do not cover all types of data loss. The ARC utility provides data protection for situations such as:

- Loss of an AMP for non-fallback tables
- Loss of multiple AMPs in the same cluster
- Failed batch processes
- Accidentally dropped tables, views, or macros
- Miscellaneous user errors
- Disaster recovery

Processing Phases

Archive or recovery jobs always operate in two phases: the Dictionary phase and Data phase. The steps of each phase are described in the following section.

Note: The archive process is intensive. You may want to create a user just for archive activities so that you can log on as the administrative user to perform other actions while archive is running.

Dictionary Phase Steps

1. Allocate an event number.
2. Issue a BEGIN TRANSACTION statement.
3. Resolve object name.
4. Check privileges.
5. Place utility locks on Data Dictionary rows and data rows.

6. Delete existing tables prior to RESTORE.
7. Issue an END TRANSACTION statement.

Data Phase Steps

1. Issue a BEGIN TRANSACTION statement.
2. Insert rows into RCEVENT and RCCONFIG.
3. Perform the operation.
4. Update RCEVENT.
5. Release locks (if user specified).
6. Issue an END TRANSACTION statement.

Session Control

To use the ARC utility, you must log on with the LOGON statement to start a session before you can execute other ARC statements. The user ID with which you log on has to have privileges for the ARC statements that you want to use.

LOGON/LOGOFF Statements

Statement	Description
LOGON	Specifies the name of the Teradata machine that ARC should connect to, as well as the username and password that should be used.
LOGOFF	• Ends all Teradata sessions logged on by the task. • Terminates the utility.

To release a lock held by another user, you must specify Override and have DROP privileges on the underlying objects.

Optimum Number of Sessions

The optimum number of sessions for archive and recovery operations are listed in the following table.

Operation	Optimum Number of Sessions Per AMP
archive	one session
restore	two sessions
recover	two sessions

Multiple Sessions

You can specify the number of archive or recover sessions with which to work, or use the default. To set the number, use the SESSIONS runtime parameter.

The number of sessions to use can vary based on a number of factors. Several are described below. Two or three sessions per AMP is a good starting point.

The vprocs use the sessions as follows:

- Teradata assigns each session to a vproc. All sessions stay with that vproc until all required data is archived. Then it will be moved to another vproc if necessary.
- Archive attempts to build blocks from each vproc in turn. The blocks are composed of complete database blocks.
- Data blocks from different vprocs are never mixed within the same archive block.

If fewer than one session per vproc is specified for the archive:

- For vproc groups, archive/recovery will archive blocks from each group with each vproc completed before the next starts.
- In this case, a large number of sessions allocated to recovery will not help recovery performance.

For larger configurations, say over 100 AMPs, specifying one session per AMP will not increase performance because of other limiting components.

In this case, for maximum throughput, cluster level operation is recommended with one session per AMP for involved AMPs.

For example, if the system has 50 clusters with 4 AMPs each, you can partition it into two jobs with 25 clusters each and 100 sessions per job provided that your site has two (or more) tape drives available and enough host resources to run two jobs in parallel.

Recommended Archive Routines

Requirements for how often and when to archive data tables vary from site to site. For tables that change on a regular basis, the best practice is to archive data once a week. The following section provides suggestions for archiving on a daily and weekly basis.

Daily Archive Routine

Perform the following procedure on a daily basis:

1. Submit a CHECKPOINT WITH SAVE statement for each journal table. This appends any changes stored in the active journal subtable to the saved journal table, and initiates a new active journal subtable.
2. Archive each current journal.
3. Delete the saved journal subtable from the saved journal.

Weekly Archive Routine

Teradata recommends that you submit an all-AMPs ARCHIVE of all data tables that change often, at least on a weekly basis. However, you may need to adjust how often you perform routine archiving based on the amount of data and based on how often your data changes. For more information on the syntax and usage of the ARCHIVE statement, see *Teradata Archive/Recovery Utility Reference*.

Using ARC With Modified PIs or Partitioning

If you plan to modify PIs with ALTER TABLE (instead of recreating the table) or to use partitioning, consider the following:

- You cannot copy an archive containing tables with PPIs to a Teradata Database that is earlier than V2R5.0.
- An archive from a V2R6.0 release or later cannot be restored or copied to a release prior to V2R6.0. However, an archive from a release prior to V2R6.0 may be restored or copied to V2R6.0 or later. For V2R6.0 and on, archives may be for selected partitions by specifying the PARTITIONS WHERE option.
- After completion of a copy or restore involving tables with PPIs, make sure the table headers and partitions are correct by running one of the following:
  - LEVEL 3 command of the CheckTable Utility.
  - REVALIDATE PRIMARY INDEX option of the SQL ALTER TABLE statement checks the primary index for a PPI table. REVALIDATE PRIMARY INDEX should be used when restoring a PPI table that needs to revalidate the primary index (for example, when copying a PPI table to a different machine).

    Note: REVALIDATE PRIMARY INDEX is an SQL option that should not be confused with the ARC statement REVALIDATE REFERENCES FOR. The REVALIDATE

PRIMARY INDEX command regenerates the table headers and optionally corrects partition numbers for a specified table with a PPI. The REVALIDATE REFERENCE FOR statement, on the other hand, is an ARC statement that corrects header indexes for referential integrity (RI) references that have been marked as inconsistent. For more information on RI and reference indexes, see *Teradata Archive/Recovery Utility Reference*.

- The table version number changes when REVALIDATE PRIMARY INDEX is used, or you use MODIFY PRIMARY INDEX to alter the partitioning, uniqueness, or column list of any PI.

  A new version or other structural change means you cannot perform the following operations on those tables:

  - Cluster restore
  - Single AMP restore
  - Permanent journal ROLLFORWARD or ROLLBACK

  Note: If ARC encounters a table with a changed version during a ROLLBACK or ROLLFORWARD operation, it stops the operation and places an error message in the output listing.

In some cases, however, a restore or copy of selected partitions is allowed even if the version number changes.

For more information on primary indexes and partitioning, see "Primary Indexes (PI)" on page 58 and "Solving PPI and RI Validation Errors" on page 411.

Restoring or copying selected partitions have the following restrictions:

- Disables RESTORE FALLBACK option
- Marks RI references as inconsistent
- Requires that options on database-level objects are not transferred to table-level objects. If you wish to apply options on the table-level, you must do so on the table-level.

For more information on PPI, see *Teradata Archive/Recovery Utility Reference*.

Archive-Related Statements

The ARC utility offers several statements to perform archive tasks. The following table lists some of these statements.

Archive-related Statement	Function
ARCHIVE	Copies a database or table to some type of portable media. Use this command to extract data from the Teradata Database.
CHECKPOINT	Places a bookmark in a permanent journal or prepares a permanent journal for archiving.

Archive-related Statement	Function
LOGGING ONLINE ARCHIVE ON/ OFF	Archive a table or database when update transactions are still concurrently occurring.
RELEASE LOCK	Releases host utility locks on databases or tables.

For more information on ARC utility commands, see *Teradata Archive/Recovery Utility Reference*.

Specifying What to Archive

Use the ARCHIVE statement to specify what you want to archive. You can archive the following:

- A single database or table
- Multiple databases or tables from all AMPs
- Multiple databases or tables from specific AMPs
- Multiple databases or tables from a specific cluster of AMPs
- All databases
- All databases in the system except for specific databases
- All databases in the system except for specific tables

The default is ALL AMPS (which is also known as a dictionary archive) and the system performs an ALL AMP-level archive unless you specify a processor or cluster archive.

You can archive specific partitions of a PPI table using the PARTITIONS WHERE option. This is useful for when you only want to back up a subset of the data rather than all of it. For example, if you do not want to perform a full-table backup every time you add or drop a secondary index or if you want to archive only the recent partitions and not have to save already archived data.

For more information, see "Types of Archives" on page 261.

Archiving Tables by Type

The ARCHIVE statement can only back up one type of table at a time: dictionary, data, no fallback, or journal. You must submit separate statements to archive each.

The following table describes the option you specify in the ARCHIVE statement.

Archive Type	Description
DICTIONARY TABLES	Backs up Data Dictionary rows that describe the databases or tables archived during a cluster- level archive. If you archive a database, the archive includes table, view, and macro definitions. If you archive a table, back up only includes table definition rows. Data dictionary information for permanent journals is not included.
DATA TABLES	Archives fallback and no-fallback tables, views, macros, stored procedures, triggers, and user-defined functions when you archive from ALL AMPs or clusters of AMPs. For a list of tables archived by an ARCHIVE DATA TABLES (DBC) statement, see *Teradata Archive/Recovery Utility Reference*.
JOURNAL TABLES	Archives the dictionary rows and selected contents of the journal tables.
NO FALLBACK TABLES	Run this archive type only to back up no fallback tables on an AMP that was down during a DATA TABLE archive. It completes the previous ALL AMP or cluster archive.

Archiving by Clusters of AMPs

You can run a cluster archive in parallel, or schedule it to run over several days. It may be faster to restore a single AMP, since the system has fewer tapes to scan to recover lost data.

In general, cluster archiving improves the archive and recovery performance of very large tables. In addition, it simplifies the restore process of non-fallback tables for a specific AMP.

Cluster archives have the following rules:

- You cannot create a cluster archive of journal tables.
- You cannot archive selected partitions.
- You cannot set up cluster archives when you are archiving DBC.
- You can partially archive non-fallback tables if an AMP is offline.
- Fallback tables are always archived, regardless of the configuration.
- Single-processor archives are only used to complete the archive of no-fallback tables after a processor is restored to service.
- A cluster archive does not contain any dictionary information.
  - You must perform a DICTIONARY TABLE archive before you run a cluster archive for the first time, because the DBC database is automatically excluded from this kind of archive operation.
  - You must run the dictionary table archive again any time there is a change in the structure of the tables in the cluster archive.
- Perform a dictionary table archive as part of the archive job set for each cluster archive run. This avoids restore issues that can occur if DDL changes occur between a dictionary table archive and the actual cluster archive.

Archiving After Recovering a Down AMP

When an AMP is offline during an all-AMP archive, non-fallback tables may only be partially archived. You need to:

- Issue the ARCHIVE statement after the AMP comes back online.
  This AMP was down during the all-AMP archive.
- Perform a single-AMP backup.
- Choose NO FALLBACK TABLES, to obtain a complete backup. (Fallback tables are always completely archived even if an AMP is down, because there is either a primary or fallback copy of the data on another AMP.)

Archiving Selected Partitions

You can select partitions for archive and restore on a PPI table by using the PARTITIONS WHERE clause in the ARCHIVE script. Incremental archiving is possible if you base the partitioning expression on a date field indicating when a row was inserted or updated. Using partitions in this manner can significantly reduce the time required for doing archives because only the recently updated partitions are archived and not the entire table.

For example, to archive all rows in the TransactionHistory table (in the SYSDBA database) for the July 2002 partition:

```
ARCHIVE DATA TABLES
(SYSDBA.TransactionHistory)
(PARTITIONS WHERE         (! TransactionDate BETWEEN DATE'2002-07-01'
                                 AND DATE '2002-07-31' !)),

RELEASE LOCK, FILE=ARCHIVE;
```

Or, to archive all tables in the SYSDBA database, except for TransactionHistory, where only the rows for the September 2003 partition are archived:

```
ARCHIVE DATA TABLES    (SYSDBA)
(EXCLUDE TABLES (TransactionHistory)),
(SYSDBA.TransactionHistory)
(PARTITIONS WHERE         (! TransactionDate = DATE '2003-09-01' !)),
RELEASE LOCK, FILE=ARCHIVE;
```

Note: Specifying one day in the month selects the entire partition. This is because when a part of a partition qualifies for the condition, the entire partition is selected.

For detailed usage information, restrictions, and examples of using PPI for archiving, see *Teradata Archive/Recovery Utility Reference*.

Archive Options

The ARCHIVE statement includes several options. The following table lists some of them.

Archive Option	Description
ABORT	Keyword to abort an all-AMPs or cluster operation for when you archive non-fallback tables or single image journal tables and an AMP goes offline during the operation.
EXCLUDE	Keyword to prevent specific databases from being archived.
EXCLUDE TABLES	Keyword to prevent specific tables from being archived.
INDEXES	For all-AMP archives only, this option specifies to include secondary indexes with the archive. You will need more time and media to archive objects with their secondary indexes.
NONEMPTY DATABASES	Instructs the ARC utility to exclude users/databases without tables, views, or macros from the archive.
PARTITIONS WHERE	Archive selected partitions of a PPI table rather than the entire table.
ONLINE	Archive rows that have update, insert, or delete operations also ongoing on those rows by using the online archive option. For more information, see "Using Online Archive" on page 259.
RELEASE LOCK	Automatically releases utility locks when the operation completes successfully.
USE GROUP READ LOCK	Permits you to archive as transactions update locked rows. You must define after image journaling for the table during the time the archive is taking place.

For a more exhaustive list of options available with the ARCHIVE statement, see *Teradata Archive/Recovery Utility Reference*.

The Indexes Option

If you archive tables on all AMPs, you can also archive the secondary indexes using the INDEXES option of the ARCHIVE statement. Archive operations automatically archive primary indexes, but do not automatically archive secondary indexes. The INDEXES option enables you to archive secondary indexes as part of the archive process only if you specify ARCHIVE DATA TABLES ALL AMP.

The INDEXES option archives both unique and nonunique secondary indexes on all data tables. However, if an AMP is offline, the utility only archives unique secondary indexes on fallback tables and ignores the nonunique indexes. In addition, it does not archive *any* secondary indexes for non-fallback tables. For this option to be the most effective, it is best to use it when all vprocs are online.

The reverse process is true for restoring data that was archived with the INDEXES option. All indexes are restored if all AMPs are online. If an AMP is down, only unique secondary indexes are restored and only for fallback tables. Nonunique secondary indexes are not restored. No indexes are restored for non-fallback tables.

For more information, see *Teradata Archive/Recovery Utility Reference*.

The Group Read Lock Option

The group read lock option allows an archive operation to proceed while you and other users make changes to the table.

Requirements and restrictions include the following:

- The backup must be an all-AMP or cluster-level archive.
- You cannot archive the DBC database with GROUP READ LOCK.
- The table must have after-image journal and the journal must be archived to complete the backup.

The ARC utility places a read lock on tables during archive operations that prevents users from updating a table during the process. The archive must be complete and the lock released before processing resumes to the table.

You can use the keyword GROUP with READ LOCK to circumvent this limitation. With GROUP, the read lock is placed at the row level, as follows:

1. The utility places an access lock on the entire table.
2. A group of table rows (about 64000 bytes) are locked for read.

   For locks that span partitions, locking is a little bit different. The lock is placed on all rows on that AMP for that partition, but the system would only archive about 64000 bytes.
3. The locked rows are archived.
4. The lock is released on that group of rows.
5. Another group of rows is locked.

The access lock prevents anyone from placing an exclusive lock on the data table while the archive is in process. By placing a read lock that disables writing on a small group of rows within the table, users can continue to make updates directly to the rows not being archived.

For rows that are locked, changes are delayed until the lock is released. Note that the lock is held for a short duration and released as soon as the group of rows is archived.

In either case, the changes are written to the after-image journal. The after-image journal must be backed up to have a complete archive of all data.

Using Online Archive

Archive a table or database when update transactions are still concurrently occurring by using the ONLINE option. Online archiving allows you to archive without using the after image journal option and without using the ARCHIVE statement with GROUP READ LOCK option.

The system creates and maintains a log for each table. It then captures all changes on the table in the log. The log of all changed rows is archived as a part of the archive process and will be restored and applied to undo the changes.

For example, to perform an all-AMP online backup of a set of database and tables, submit the following:

```
ARCHIVE DATA TABLES (DB1),(DB2,T1), …,
ONLINE,
RELEASE LOCK,
FILE=ARCHIVE;
```

You can perform the following types of archives with the online archive option:

- Database archive
- Table archive
- Cluster archive
- Archive with down AMP (only Fallback-protected tables are supported)

The maximum number of tables on which you can activate online archive logging is 10,000.

The following statements and actions are not allowed[1] on a table with online archiving enabled:

- ALTER TABLE
- CREATE HASH INDEX
- CREATE INDEX
- CREATE JOIN INDEX
- DROP HASH INDEX
- DROP INDEX
- DROP JOIN INDEX
- CREATE new objects in the database[2]
- Reconfig
- FastLoad

Caution: Objects (table, view, macro, and so on) can be dropped after logging online archive is turned on for a database.

The tables specified in the same LOGGING ONLINE ARCHIVE ON statement will be archived at a transactionally consistent point. However, if some tables are initiated from different LOGGING ONLINE ARCHIVE ON statements, there is no guarantee that tables from two LOGGING ONLINE ARCHIVE ON statements will be archived at a transactionally consistent point.

[1] Use the LOGGING ONLINE ARCHIVE OFF statement to stop the online archive logging activity if you need to submit any of the restricted statements or perform the restricted actions. This statement should contain the same list of tables/databases specified in a previous LOGGING ONLINE ARCHIVE ON statement and will delete the log subtables associated with the object.

[2] This only applies if the online logging is activated on the database. If only a specific table within a database has been online logging activated, then you can still create new objects within that database.

Locks

A table level online archive does not use any locks placed on the tables at all. For a database level online archive, the database is protected under a utility level database access lock.

The database access locks prevents any exclusive lock operations from running on objects within the database. These operations include create and dropping tables, macros, views, UDFs, and so on. These operations, including operations that delete and change data such as a table rebuild, cannot occur while the database is at a database level online archive.

A table level online archive does not use locks so that there is a possibility that the table could be updated and changed. However, in order to properly capture the change images for the rows, The system does not allow DROP TABLE, RESTORE TABLE, and TABLE REBUILD statements during an archive operation.

Types of Archives

The following types of archives back up data on removable media. The information in the archive file depends on the archive type, the table protection used, and your Teradata Database configuration.

Note: You can fully restore database DBC only to an empty Teradata Database.

Archive Type	Table Protection Type	Data Included
All AMP database	Fallback	• Primary data rows and Secondary Indexes (SIs) from all the tables in the database(s) • Data Dictionary rows of the tables/macros/views/functions/stored procedures • All table/macro/view/stored procedure/triggers/UDF information • Table structure information
	No fallback	• Available data rows • Data Dictionary rows of the tables/macros/views/functions/stored procedures/triggers/UDF • All table/macro/view/stored procedures/triggers/UDF information • Table structure information Note that no SIs are included if the AMP is down.
All AMP table	Fallback	• Primary data rows • Secondary Indexes (SIs) • All dictionary information for the table • All table, column, and index definitions
	No fallback	• Available data rows • All dictionary information for the table • All table, column, and index definitions Note that no SIs are included if the AMP is down.

Archive Type	Table Protection Type	Data Included
Specific AMP	No fallback	• Available data rows from the table(s) within the database(s) • Table structure information Note that: • No dictionary rows are included. This information is included in the Dictionary archive. • No SIs are included.
Specific cluster	Fallback or no fallback	• Available data rows from the table(s) within the database(s) • Table structure information Note that: • No dictionary rows are included. This information is included in the Dictionary archive. • No SIs are included.
Data Dictionary		Dictionary rows for tables/macros/views/stored procedures/triggers/UDFs. For tables: • DBC.TVM • DBC.TVFields • DBC.Indexes • DBC.IndexNames Note that PJ information is not included.

Archiving the Data Dictionary

The Data Dictionary tables are maintained in the default database associated with system user DBC. This means that as they grow, they consume DBC PERM space, which is space also required by the TJ and other internal recovery journals. To protect your system software, you should archive the DBC database when:

- A high volume of DDL statements has significantly increased the quantity of definitions for data tables, views, macros, indexes, triggers, stored procedures, roles, or profiles since your last DBC archive.
- You plan to purge some dictionary logs that have grown very large due to a high volume of recorded activities, queries, or statistics.
- You plan to upgrade your Teradata Database or migrate to a different server platform.

  Some dictionary tables, such as DBQLRuleTbl, are not archived automatically. If you want to retain the contents through a migration, which requires a system initialization, you might need to recreate the rows as follows:

  - Maintain all statements that control rules, for example BEGIN/END QUERY LOGGING, in a BTEQ executable (script) file.

- After the migration has completed (and after the DIP utility and the ARC Restore of data and dictionary tables have been run), execute the BTEQ script to recreate the contents of the table.

Note: When archiving partitions of the Data Dictionary you cannot specify the ALL PARTITIONS option.

For more information about archiving the Data Dictionary tables and a full list of the tables archived by the system, see *Teradata Archive/Recovery Utility Reference*.

Restoring Data

Introduction

A restore operation transfers database information from archive files backed up on portable storage media to all AMPs, clusters of AMPs, or specified AMPs.

You can restore archived data tables to the Teradata Database if the Data Dictionary contains a definition of the object you want to restore.

For example, if the object is a database, that database must be defined in the dictionary. Or, if the object is a table, that table must be defined in the dictionary. You cannot restore objects not defined in the Data Dictionary.

A dictionary table archive contains all table, view, macro and trigger definitions in the database, and dictionary entries for stored procedures and user-defined functions. A restore of a dictionary archive restores the definitions of all data tables, views, macros, triggers, and stored procedures. However, it does not restore any data.

Caution: You cannot restore a database with join indexes or a table referenced by a join index. You must first drop join indexes before restoring. If you attempt to restore with join indexes still defined, you will get an error message.

Restore-Related Statements

The ARC utility provides several recovery control statements you use during restore-related operations. Each command is described in the following table.

Restore-Related Statement	Function
ANALYZE	Reads an archive tape to display information about its contents.
BUILD	Builds secondary indexes for fallback and non-fallback tables. It also builds fallback rows for fallback tables, and can build journal tables by sorting the change images.
COPY	Restores a copy of an archived file to a specified Teradata Database.

Restore-Related Statement	Function
DELETE DATABASE	Deletes data tables, views, and macros from a database. Does not remove journal tables.
DELETE JOURNAL	Removes a journal subtable from the Teradata Database.
RELEASE LOCK	Releases host utility locks from specific databases or tables.
RESTORE	Restores a database or table from an archive file to specified AMPs.
REVALIDATE REFERENCES FOR	Validates inconsistent restraints against a target table, thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables. **Note:** The REVALIDATE REFERENCES statement does not validate soft RI (tables with NO CHECK option.)

ANALYZE Statement

The ANALYZE statement reads data from an archive tape and displays information about tape contents. When you invoke the statement, you can choose a specific database or a range of databases from which to display information. You can analyze:

- Time and date of the archive operation
- The archive level: all-AMPs, clusters of AMPs, or specific AMPs
- The name of each database, data table, journal table, view, and macro in each database and the fallback status of the tables. Information appears only if you use the keyword LONG with the DISPLAY option.
- display diagnostic information related to an archive of selected partitions. You can use this statement to verify what partitions were archived.

This information helps you if you are trying to restore a specific database instead of the entire archive set. This statement does not require a prior logon.

BUILD Statement

The BUILD statement recreates unique and nonunique secondary indexes on non-fallback and fallback tables and generates journal tables by sorting the change images.

You must rebuild indexes for non-fallback tables after a restore operation if any of the following situations occur:

- An AMP is offline during a dump or restore
- The restore operation is not an all-AMP restore
- The archive did not include the INDEXES option
- The restore included the NO BUILD option

Example

The following example illustrates the BUILD statement. The example builds unique and nonunique secondary indexes for all tables on the archive tape. The release lock option removes the utility lock after successful completion of the build operation.

```
BUILD DATA TABLES (Personnel) ALL
, RELEASE LOCK;
```

COPY Statement

Use the COPY statement to recreate tables and databases that have been dropped or to restore them to the same system or to a different system. The COPY statement:

- Uses an archived file to recreate tables and databases that have been dropped.
- Copies archived files to a different system.
- Can replace the creator name of the copied objects with the name of the current user (the user submitting the COPY command). Otherwise, the name of the original creator is retained.
- Can copy selected partitions if specified with the PARTITIONS WHERE option.

You can use the COPY statement to copy an object:

- That has been dropped back into the original system.
- From one system to another.
- Back to the same system.

Example 1

Example 1 copies an archived data table called Personnel.Department from an archive file to a different Teradata Database system.

```
COPY DATA TABLE (Personnel.Department)
,FILE=ARCHIVE;
```

Example 2

Example 2 copies the same archived data table from its old database, OldPersonnel, to a new database. The no fallback option indicates that the new table is to be non fallback on the receiving system even if it was fallback on the original one. The NO JOURNAL option indicates that you do not want permanent journaling on this table in the receiving database.

```
COPY DATA TABLE (Personnel.Department)
(FROM (OldPersonnel), NO JOURNAL, NO FALLBACK)
,FILE=ARCHIVE ;
```

RELEASE LOCK Statement

The ARC utility places locks on database objects while it performs archive and restore activities. These locks are referred to as host utility-level locks. The ARC utility does not automatically release these locks upon successful completion of an ARC command. In fact, these locks remain intact even when an AMP goes down and comes back online. You must submit the RELEASE LOCK statement to remove the locks.

Not everyone can issue the release lock statement. You must have either the ARCHIVE or the RESTORE privilege on the locked object or be the owner of the locked object.

You may submit the RELEASE LOCK option at the same time you issue ARCHIVE, ROLLBACK, ROLLFORWARD, RESTORE, and BUILD commands. This accomplishes the same purpose as issuing the RELEASE LOCK statement.

RESTORE Statement

Use the RESTORE statement to replace tables or databases from an archive. The RESTORE statement allows you to replace database objects from an archive tape to the same system or to another system. Teradata Database features the four types of RESTORE or RECOVER operations described in the following table.

RESTORE Option	Function
ABORT	This option causes an all-AMP restore to abort with error messages if an AMP is offline and the restore includes a non-fallback table. It does not affect a specific AMP restore.
DATA TABLES	The DATA option restores fallback, non fallback, or both types of data tables to all AMPs or clusters of AMPs.
DICTIONARY TABLES	The DICTIONARY option restores Data Dictionary rows that describe the databases or tables archived during a cluster-level archive. If you restore a database, the table, view, and macro definitions from the Data Dictionary are included. If you restore a table, only table definition rows are included.
EXCLUDE TABLES	The EXCLUDE TABLES option allows you to delete all objects in the target database excluding the indicated tables. This ensures that prior increments based on selected partitions that you have accumulated in the target database are not disturbed while you replace everything else with the most recent snapshot. To use EXCLUDE TABLES option on a restore, you must have either the SELECT privilege on the view DBC.TABLES or database DBC.
JOURNAL TABLES	This option restores an archived journal for subsequent use in a roll operation.
NO FALLBACK TABLES	Use the no fallback option to restore a single processor.
NO BUILD	Prevents secondary indexes on non-fallback tables from being restored or built. On fallback tables, it prevents the creation of secondary indexes and fallback table rows.

RESTORE Option	Function
PARTITIONS WHERE	Allows you to restore specified partitions. Restores of selected partitions are not supported for a system with a different hash function than the one used for archive. **Note:** Restoring selected partitions that contain one or more LOB columns is not supported. To restore the selected partitions from the backup in either of the above two cases, do the following: • Make a full-table copy of the backup to the system to a staging table. • Delete the rows from the selected partitions as needed in the target table. • Insert-select from the staging table into the target table • Delete the rows in the staging table or drop the staging table.
RELEASE LOCK	Causes ARC to automatically release the utility locks when a restore completes successfully.
RESTORE FALLBACK	Applies only to data table restored on fallback tables, and allows the utility to restart the restore without returning to the first row (in the event of a processor failure).

Example 1

This example restores all databases from the entire system when all AMPs are online. The restore type is data and the restore object is all databases belonging to user Payroll. Because there is no mention of any restore levels, such as a specific AMP number, the system assumes all AMPs.

The release lock option removes the utility lock after completing the restore operation. The name of the archive file is MYDATA.

```
Restore ALL AMPs with ALL AMPs online:
LOGON Sysdba, xxxxxxxx ;
RESTORE DATA TABLES (Payroll) ALL,
RELEASE LOCK,
FILE=MYDATA;
LOGOFF ;
```

Example 2

This example has a narrower scope than Example 1. This statement is only restoring non fallback tables on AMP 5. The administrator has already performed an all-AMPs restore on the rest of the system. The restore excludes user Payroll.

The release lock option removes the utility lock after completion of the restore operation. The archive filename is MYDATA2.

Any databases or users created since the archive of the dictionary or any table, view, or macro created since the archive of a database, is dropped when you restore the DBC database or a user database.

To perform a restore on AMP that was offline:

```
LOGON Sysdba, xxxxxxxx ;
RESTORE NO FALLBACK TABLES (Payroll) ALL,
AMP=5
RELEASE LOCK,
FILE=MYDATA2;
LOGOFF ;
```

Example 3

Restore all data for all tables in database SYSDBA, including all partitions archived for table TransactionHistory:

```
RESTORE DATA TABLES
(SYSDBA)
(EXCLUDE TABLES (TransactionHistory)),
(SYSDBA.TransactionHistory)
(ALL PARTITIONS),
RELEASE LOCK,
FILE=ARCHIVE;
```

Note: Archive for SYSDBA.TransactionHistory must be of “well-defined” selected partitions.

There are additional conditions you can define to further specify the PARITIONS WHERE option. For example, you can restore certain rows to an error table.

For more information, see *Teradata Archive/Recovery Utility Reference*.

Changing the State of RI Tables with REVALIDATE REFERENCES FOR

When either a parent or child table is restored, the reference is marked inconsistent in the database dictionary definitions. As a result, the system does not allow UPDATE, INSERT, or DELETE statements on such tables.

The REVALIDATE REFERENCES FOR statement validates the inconsistencies, thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables. The functions performed by this statement include:

- Validates the inconsistent reference index on the target table
- Creates an error table
- Inserts into the error table rows that fail the referential constraint specified by the reference index

If inconsistent restraints remain after you execute the statement, you can use the ALTER TABLE DROP INCONSISTENT REFERENCES statement to remove them. To use the REVALIDATE REFERENCES FOR statement, the username you have specified in the LOGON statement must have one of the following privileges (for more details, see “Using Referential Integrity” on page 240):

- RESTORE privileges on the table you are re-validating
- Implicit privileges on the database or table

Note: The REVALIDATE REFERENCES statement does not validate soft RI (tables with NO CHECK option.)

Restoring After a Down AMP

Assume the following:

- Your system uses RAID 5.
- On Monday, you archive journals X, Y, and Z.
- On Tuesday, you archive journals X, Y, and Z.
- On Wednesday, you archive journals X, Y and Z and archive tables X, Y and Z.
- On Thursday, two disks fail in a rank on AMP 3.
- Teradata Database includes tables with the following kind of protection:
  - Fallback with dual before- and after-image (Table X)
  - Dual before- and after-image (Table Y)
  - Single before-image and single after-image (Table Z)

To recover after repairing failed disk hardware, perform the following:

1. Rebuild AMP 3. Use Vproc Manager to format the new disks and build the Teradata Database file system. See "Vproc Manager (vprocmanager)" in *Utilities*.
2. Restart the database. For more information, see "Restarts of the Teradata Database" on page 285.
3. Restart processing brings AMP3 online.
4. Use the Table Rebuild utility on AMP 3 when it is up. See "Table Rebuild (rebuild)" in *Utilities*.

   After the disks are configured, the status of each table is now as follows:

Table	Status
X	All data rows and all before- and after-images are restored.
Y	The PJ tables are restored, but the primary data table is still missing. You need to perform a restore procedure to restore table Y. See "Restoring Dual Before- and After-Image Protected Tables" on page 269.
Z	The PJ tables are not restored and the primary data table is missing. You need to restore table Z. For the procedure, see "Restoring Single Before- and After-Image Protected Tables" on page 270.

Restoring Dual Before- and After-Image Protected Tables

To fully restore Table Y, follow this procedure:

1. Perform a single-AMP RESTORE of AMP 3 using the Tuesday dump of Table Y.

   Do *not* release read locks.
2. Restore the Wednesday DUMP of Journal Y for the AMP 3.

3. Perform a single-AMP ROLLFOWARD on AMP 3 with the restored journal from Table Y. This replaces the existing rows in Table Y with any after-images made since the last backup on Tuesday.
4. Submit the DELETE JOURNAL statement to delete the restored Journal Y. This deletes all stored images from the restored journal.
5. Perform a single-AMP ROLLFOWARD on AMP 3 with the CURRENT journal from Table Y. This replaces the existing rows in Table Y with any after-images stored in active or saved PJ subtables.
6. Release all utility locks.

Restoring Single Before- and After-Image Protected Tables

To restore Table Z, perform the procedure detailed below:

Note: This procedure restores the data table, but does not restore the PJ tables.

1. Perform a single-AMP RESTORE of AMP 3 using the Wednesday DUMP of table Z. This restores all of the data rows stored in the archive file from Table Z.

   You do not need to restore the journal tables for Table Z, since you performed a complete backup of Table Z on the same day as the journal archive. All the changes through Wednesday are in the archive of the entire table.

   Do *not* release the utility locks.
2. Perform a single-AMP ROLLFORWARD on AMP 3 using the CURRENT journal from Table Z.

   This replaces existing table rows with any after-change images stored in the active or saved PJ subtables. Any changes in the current journal would have occurred on Thursday before the disk failure.
3. Perform an all-AMPs DUMP of Table Z to protect against a second disk failure in the same cluster. You cannot restore the journal for AMP 3 because you did not specify dual images for Table Z; therefore, another disk failure in the cluster leaves data unrecoverable.
4. Submit a CHECKPOINT WITH SAVE statement.

   This action:

   - Moves any stored images from the active subtable to the saved subtable of the current journal
   - Initiates the active subtable.
5. Submit a DELETE SAVED JOURNAL statement.

   This erases the contents of the saved subtable, since the contents are no longer needed.
6. Release the utility locks with the RELEASE LOCKS command.

Using FastLoad to Restore Data

The FastLoad utility can restore archived information to disk. To do this, instead of archiving to tape, use the BTEQ EXPORT command or FastExport to store the information in a host file. FastLoad can quickly load large amounts of data to an empty table on Teradata Database but loads into only one table per job. To load data into more than one table, submit multiple FastLoad jobs.

FastLoad Steps to Restore a Table

FastLoad operates only on tables with no secondary indexes, join indexes, or triggers. You have to recreate these objects when the FastLoad completes. To restore a table, FastLoad does the following:

1. FastLoad uses a single session to send the INSERT statement to the PEs and AMPs.
2. Multiple sessions are then used to facilitate sending rows to the AMPs.
3. Upon receipt, each AMP hashes each record and redistributes it over the BYNET. This is done in parallel.
4. The receiving AMP then writes these rows as unsorted blocks directly to the target table.
5. When loading completes, each AMP sorts the target table, puts the rows into blocks, and writes the blocks to disk.
6. Fallback rows are generated if required.

Recovery Steps

Recovering data to the same configuration means the data blocks recovered to the AMP are already in the appropriate format. Recovering to a different configuration uses a different algorithm. The rows are collected and during the BUILD phase, the rows are sorted.

The ARC utility is great for recovering a very large number of objects and can restore an entire machine with one command. Using FastLoad is an option for also quickly loading recovered data, but can only load into empty tables and load only one table per job.

For more information on how to use FastLoad, see *Teradata FastLoad Reference*.

Recovering Data

Some important concepts regarding data recovery operations are:

- As with archive and restore operations, you use the ARC utility with permanent journals (PJs) for data recovery operations.
- The CHECKPOINT statement indicates a recovery point in a journal.
- The CHECKPOINT WITH SAVE statement saves stored images before a row marker in an active subtable and appends them to the saved subtable.

- ROLLBACK or ROLLFORWARD operations, which involve PJs, can use either current journals (active and saved subtables) or restored journals (restored subtable). ROLLBACK commands help you recover from one or more transaction errors and reverses changes made to a database or table.
- ROLLFORWARD commands help you recover from hardware errors. These commands replace existing row data with after-change images.
- DELETE JOURNAL command erases the contents of either the restored subtable or the saved subtable in the PJ.
- There are several recovery control system views that contain information about ARC utility events.
- You can cancel rollback if you decide it is quicker to restore the tables instead.

Data Recovery Using Roll Operations

The restore statement allows you to move information from archive files back to Teradata Database. The restore operation can restore data or journal tables. When you restore a journal table, the system restores the information to a permanent journal subtable. Before you can use the tables, you must perform a rollback or rollforward operation to move the journal tables back to the data tables.

Roll operations can use either the current journal or the restored journal. If you specify the current journal, then the ARC utility uses information stored in both the active and saved subtables.

A permanent journal is checkpoint-oriented rather than transaction-oriented. The goal of the journals is to return existing data tables to some previous or subsequent checkpoint. For example, if a batch program corrupted existing data, the rollback operation would return the data to a checkpoint prior to the running of the batch job.

A rollforward operation might occur after an all-AMP restore. After you move the data and journal archive files back to the database, the data tables would only include changes committed since the last full backup. Any intermediate changes would reside in the journal tables. The rollforward operation would replace the existing data with changes from the journal table.

In summary:

- The RESTORE function copies journal archive files to the restored subtable of the PJ.
- ROLLBACK and ROLLFORWARD statements apply journal table contents to data tables.
- Roll operations can use either Current journal (active and saved subtables) or Restored journal (restored subtable).

CHECKPOINT Statement

Use the CHECKPOINT statement to indicate a recovery point in the Journal. The CHECKPOINT statement places a marker row after the most recent change image row in the active subtable of a permanent journal.

Teradata Database assigns an event number to the marker row and returns the number in response. You may assign a name to the CHECKPOINT command rather than use the event number in subsequent ARC activities.

The following table describes the options to the CHECKPOINT statement.

CHECKPOINT Option	Description
NAMED *chkptname*	Checkpoint names may be up to 30 characters long and are not case-specific. Teradata Database always supplies an event number for each checkpoint. Use the number to reference a checkpoint if a name is not supplied. If there are duplicate checkpoint names in the journal and an event number is not specified: • Rollforward uses the first (oldest) occurrence. • Rollback uses the last (latest) occurrence.
USE LOCK	By default, the system acquires a read lock on all tables assigned to the journal being checkpointed. A checkpoint with save may optionally use an access lock. The read lock suspends update activity for all data tables that might write changes to the journal table during checkpoint. This lock provides a clean point on the journal. The access lock accepts all transactions that insert change images to the journal, but it treats them as though they were submitted after the checkpoint was written. The access lock option requires that you also use the WITH SAVE option. A checkpoint with save under an access lock is only useful for coordinating rollforward activities from the restored journal, and then from the current journal.
WITH SAVE	The WITH SAVE option logically moves the contents of the active subtables of the identified journals to the saved subtable. After you archive the saved area of the journal, you can delete this section of the current journal to make space for subsequent saved journal images. The database automatically initiates a new active subtable. You can dump the contents of the saved subtable to an archive file.

Starting Rollbacks

Use the ROLLBACK statement to recover from one or more transaction errors. To use this statement, you must define the table with a before-image journal table. The ROLLBACK is performed to a checkpoint or to the beginning of the current or restored journal.

The system uses the before images to replace any changes made to the table or database since a particular checkpoint was taken.

The ROLLBACK command helps you recover from one or more transaction errors. It reverses changes made to a database or table. To accomplish this reversal, it replaces existing data table rows with before-change images stored in a permanent journal.

The before-change images must reside in either the restored or current subtables of a permanent journal. If you choose the current subtable for rollback procedures, the database uses the contents of both the active and saved subtables.

When you use the restored subtable for rollback procedures, you need to verify that it contains the desired journal table. If it does not, submit the RESTORE JOURNAL TABLE command with the appropriate removable storage media. This process ensures that you restore the correct subtable contents. Teradata Database does not have any simple tools for looking at journal subtables to determine that they contain the desired data.

Checkpoint names need to match existing names used with a previous CHECKPOINT statement. An *eventno* is the software-supplied event number of a previous checkpoint. You can supply either checkpoint names or event numbers or both. To find the checkpoint names or event numbers, select information about the checkpoint from the DBC.Events view.

If there are duplicate checkpoint names in the journal and an event number is not supplied, rollback stops at the first one encountered with a matching name.

The following illustrates a rollback procedure:

1. First, activate the ROLLBACK CURRENT JOURNAL statement to rollback any changes made since the journal table was archived. This statement rolls back the saved subtable first followed by the active subtable.
2. Next, run the RESTORE JOURNAL TABLE command to load the appropriate archive file into the restored subtable of the permanent journal.
3. Finally, submit the ROLLBACK RESTORED JOURNAL command to reverse the changes by replacing any changed rows with their before-image rows stored in the restored journal. Repeat Steps 2 and 3 as necessary.

By default, the rollback procedure automatically deletes the contents of the restored subtable after successfully completing the command. The NO DELETE option overrides the default and has the following benefits:

- Overrides automatic deletion of restored subtables, allowing you to:
  - Recover selected tables first
  - Later recover other tables that may have changes in the journal
- Is used only for restored subtables
- Is never used with current subtables

Canceling Rollbacks

With the help the Recovery Manager utility, you can cancel rollbacks that are taking a long time to complete. Canceling rollbacks allows you to make tables available more quickly *if* the time it takes to restore specific tables is less than waiting for the rollback to complete on those tables.

Rollbacks usually take up twice as long as it would take for the runtime of a job. Rollback on tables with USIs take longer. Rollbacks on tables with NUSIs take even longer. If multiple tables are involved with a complex join, rollbacks could take more than just a few hours.

However, be aware that canceling a rollback leaves the table in an unknown and unstable state if inserts or modifications were performed.

If you want to stop the rollback of a table because of an error or due to a system restart, use the Recovery Manager utility to abort the rollback, delete the table, and then reload it from backup if needed.

Note: Before you cancel a rollback, you *must first* list rollback tables. Then cancel the rollback on tables from the list. These two steps must be done in the same Recovery Manager session.

The Recovery Manager utility allows you to:

- Cancel the rollback of tables during a system restart or an aborted, online transaction.
- List the status of the tables being rolled back in the system.
- Change the priority levels for rollbacks in user-aborted transactions.

For example, a typical process might be:

1. You notice that a rollback is taking too long.
2. You identify if a large table, or perhaps several tables, can be restored more quickly than the rollback will take.
3. If canceling rollback is quicker than waiting for rollback to complete, decide which tables to cancel rollback. Submit the LIST ROLLBACK TABLES statement first and then cancel rollback on the tables in that list.
4. Immediately do a DELETE ALL and restore the table if you need that table restored. If you do not need the table, be sure to still do the DELETE ALL operation.

The following table lists some of the Recovery Manager utility commands involved with canceling rollbacks.

<table>
<tr><th>Use this RcvManager command...</th><th>To...</th></tr>
<tr><td>CANCEL ROLLBACK ON TABLE</td><td>cancel rollback on one or more tables. Use the command <i>after</i> running the LIST ROLLBACK TABLES command and use this command only when:
<ul>
<li>The rollback of a table is likely to take longer than its restoration.</li>
<li>The table, such as a temporary table, is non-critical. Some things to note:
<ul>
<li>You must have the DBC password to execute this command.</li>
<li>The priority of the cancel is the same as the priority of the rollback.</li>
<li>CANCEL ROLLBACK command is not instantaneous. The system must still read all of the Transient Journal rows for the transaction even if it does not apply most of them, or at least the rows on the table on which rollback has been cancelled.</li>
<li>You cannot cancel rollback on the referenced or referencing tables of Referential Integrity unless they are self-references.</li>
</ul></li>
</ul>
You can perform the following operations on a table for which rollback has been cancelled:
<table>
<tr><th>Valid Operations</th><th>Description</th></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Use this RcvManager command...</th><th>To...</th></tr>
<tr><td></td><td>
<table>
<tr><td>ARC DUMP</td><td>Take a dump of the table header using the ARC utility. Only the table header is dumped, not the data in the table. Then, you retain the DDL without having to recreate the table.</td></tr>
<tr><td>ARC RESTORE</td><td>Restore the table from archive using ARC. The table is made usable for update operations after it is restored from backup.</td></tr>
<tr><td>CheckTable</td><td>Skip the tables specified with the CANCEL ROLLBACK ON TABLE command.</td></tr>
<tr><td>DELETE ALL</td><td>Delete all the rows in a table and make the table valid again.</td></tr>
<tr><td>DROP TABLE</td><td>Drop the table so you can create it again.</td></tr>
<tr><td>LOCKING…with READ OVERRIDE option</td><td>Retrieve a single table for SELECT operations.</td></tr>
<tr><td>Table Rebuild</td><td>Rebuild the table headers.</td></tr>
</table>
</td></tr>
<tr><td>LIST CANCEL ROLLBACK TABLES</td><td>list all the tables for which rollback is pending cancellation as a part of an online user requested abort or during system recovery.
When rollback on a table has been cancelled, the table is marked not valid. The not valid tables <i>do not</i> appear on the list generated by the LIST CANCEL ROLLBACK TABLES command. Therefore, it is highly recommended that you perform a DELETE ALL on the table after canceling rollback on it.</td></tr>
<tr><td>LIST ROLLBACK TABLES</td><td>display all the tables that are currently undergoing rollback in the system. The list is sorted in descending order of the TJ Row Count column.
This command displays records for the tables in a transaction having more than 10,000 rows to rollback on at least one AMP. The transaction must be in abort status.
An asterisk (*) follows the tables which cannot be specified in the CANCEL ROLLBACK command because they are Referential Integrity tables.
<b>Note:</b> DBC tables and Permanent Journals are not listed.</td></tr>
<tr><td>LOCKING TABLE … FOR READ OVERRIDE</td><td>perform a single table retrieve operation on a table for which rollback has been cancelled using the CANCEL ROLLBACK ON TABLE command.
The tables for which rollback has been cancelled can be viewed through this locking modifier option.</td></tr>
<tr><td>ROLLBACK SESSION … PERFORMANCE GROUP</td><td>display or set the performance group setting of the rollback for the specified session. This command allows you to change the priority. By reducing the priority of less important rollbacks, you can increase the system resource availability for other processes in the system.
Rollback processing runs at the priority defined for the session, that is, the priority associated with the Performance Group for the session. If no priority is specified, rollback runs at RUSH priority by default.</td></tr>
</table>

For more commands, usage notes, and syntax diagrams for Recovery Manager, see *Utilities*.

Recovering Data with Rollforward

The ROLLFORWARD command helps you recover from a hardware error. It changes existing rows in data tables by replacing them with after-change images stored in a permanent journal. The after-change images must reside in either the restored or current subtables of a permanent journal.

When you use the restored subtable for rollforward procedures, you need to verify that it contains the desired journal table.

If it does not, submit the RESTORE JOURNAL TABLE command with the appropriate portable storage media. This ensures that you restore the correct subtable.

Also, before you can rollforward, you must have a backup copy of the table rows and AFTER Image journal rows since the last backup.

The following table lists descriptions of some of the options.

ROLLFORWARD Option	Description
PRIMARY DATA	During a rollforward operation, this option instructs the software to ignore secondary index and fallback row updates. A BUILD operation rebuilds the invalidated fallback copy and indexes.
TO checkpointname, eventno	Checkpoint names need to match existing names used with a previous CHECKPOINT statement. An event number is the software-supplied event number of a previous checkpoint. You can supply either one or both of these. To find the checkpoint names or event numbers, select information about the checkpoint from the DBC.Events view. If there are duplicate checkpoint names in the journal and an event number is not supplied, rollback stops when it encounters with a matching name.

The following illustrates a rollforward procedure:

1. Submit the RESTORE DATA TABLE command.
2. Submit the RESTORE JOURNAL TABLE command to load the appropriate archive files into the restored permanent journal subtable.
3. Submit the ROLLFORWARD RESTORED JOURNAL command to replace existing data table rows with their after-image rows stored in the restored journal.
4. Submit the ROLLFORWARD CURRENT JOURNAL statement to rollforward any changes made since the journal table was archived. This statement rolled forward the saved subtable first followed by the active subtable.

PRIMARY DATA Option

This option replaces only primary row images during the rollforward process. It ignores secondary index.

If you use this option with a rollforward operation, you can reduce the amount of I/O. It also improves the rollforward performance when recovering a specific AMP from disk failure.

Unique indexes are invalid when recovering a specific AMP. Always submit a BUILD statement when the rollforward command includes the PRIMARY DATA option.

Therefore, the PRIMARY DATA Option:

- Ignores secondary index rows
- Reduces amount of I/O
- Improves performance when recovering single-AMP

ROLLFORWARD Restrictions

The following table illustrates important restrictions on using the ROLLFORWARD statement.

ROLLFORWARD Restriction	Description
AMP-specific Restore	If you perform a restore operation on a specific AMP rather than on all AMPs, the ROLLFORWARD command does not permit you to use the TO CHECKPOINT NAME option. Following an AMP-specific restore, the system permits a rollforward only to the end of the journal. You must follow up the restore process with a rollforward of the entire journal table.
All-AMP Restore	When you perform an all-AMP restore, you choose whether to submit the ROLLFORWARD command with the TO CHECKPOINT NAME option, or to the end of the journal. The PRIMARY DATA option of the ROLLFORWARD statement indicates that the operation should ignore secondary index and fallback rows that will reduce the amount of I/O during rollforward. If you use this option, follow up with the BUILD statement. Use the DBC.Events view to determine event numbers and checkpoint names.

For example:

```
SELECT EventNum FROM DBC.Events WHERE CreateDate = 940819;
SELECT CheckPointName FROM DBC.Events
WHERE CreateDate = 940819;
```

DELETE JOURNAL Statement

The DELETE JOURNAL command erases the contents of either the restored subtable or the saved subtable of a permanent journal. You must have the RESTORE privilege to execute this command.

You cannot delete:

- Rows from an active journal.
- A saved subtable when all the following conditions are true:
  - A CHECKPOINT statement in the archive utilized an access lock
  - The journal is not dual image
  - One or more AMPs are offline and the saved subtable does not have a dual journal

    Note: When a journal archive has all three of the above conditions, transactions between an all-AMP archive and a single-AMP archive may not be consistent.

Recovery Control Views

Several system views contain information about ARC utility events. You can use these views for recovery control.

This view...	Provides information about ...	On table...
DBC.Association	objects (databases, users, tables, views, macros, indexes, stored procedures) that you imported from another Teradata Database system or created via the Archive/Recovery COPY statement	DBC.DBCAssociation
DBC.EventsX	archive/recovery activities, with a row (audit trail) for each archive and recovery event	DBC.RCEvent
DBC.Events_ConfigurationX	archive and recovery activities that did NOT affect all AMPs	DBC.RCConfiguration
DBC.Events_MediaX	archive and recovery activities that involved removable media	DBC.RCMedia

Association View

The Association view allows you to retrieve information about an object imported from another Teradata Database.

An existing object created with the ARC COPY command also appears in the Association view. If you later drop a copied object from its new destination, the information is deleted from the Association table and is no longer available.

The following example uses the Association view to list all tables, views, or macros that were copied into the payroll database.

The result of the query displays imported and current table names. The object column displays the current name of each table. The Source column provides the name of the original table. The event column shows the number assigned to the restore operation.

```
SELECT TRIM (DatabaseName)||’.’||TableName
(NAMED Object, FORMAT ‘X (25)’)
TRIM (Original_DatabaseName)||’.’||
Original_TableName
(NAMED Source, FORMAT ‘X(25)’)
EventNum (NAMED Event, FORMAT ‘9(5)’)
FROM DBC.Association
WHERE DatabaseName LIKE %Payroll%’
ORDER BY Object ;
```

This query returns event rows similar to the following:

```
Object                      Source                      Event
---------------------------------------------------------------
Payroll_Prod.DEPARTMENT      PAYROLL_TEST.department      00014
Payroll_Prod.DEPT            PAYROLL_TEST.dept            00014
Payroll_Prod.EMP             PAYROLL_TEST.emp             00014
Payroll_Prod.EMPLOYEE        PAYROLL_TEST.employee        00014
Payroll_Prod.NEWHIRE         PAYROLL_TEST.newhire         00014
```

Events View

The Events view tracks ARC activity. ARC inserts a new row in the Events system table each time another ARC activity begins. The Events view returns a row for each activity tracked.

The following table describes events the system creates depending on the type of object on which the activity was performed.

A row for this event type ...	Is created for each ...
Checkpoint Event Row	journal checkpointed.
Delete Event Row	journal deleted.
Dump Event Row	database or table dumped.
Restore Event Row	database or table restored.
Rollback Event Row	database or table rolled back.
Rollforward Event Row	database or table rolled forward.

The SQL statement in the following example requests a list of all ARC activity that took place March 28th.

```
SELECT EventNum
,UserName (CHAR (12))
,EventType (CHAR (12))
,DatabaseName (CHAR (15))
FROM DBC.Events
WHERE CreateDate=990328
ORDER BY EventNum ;
```

```
 EventNum         UserName           EventType      DatabaseName
 ---------        -------------      ---------      ---------------

 180              BRM                Dump           Payroll_Test
 181              RPK                Restore        Personnel_Test
```

Events_Configuration View

The Events_Configuration view contains rows for each archive activity that does not affect all AMPs in the database configuration. If the ARC command specifies all AMPs and there are one or more AMPs offline, a row is inserted in the system table for each offline AMPs. If the statement is for specific AMPs, a row is inserted for each specified and online AMPs.

The following example submits an SQL statement to find out which user did not release the utility locks on processor 2. Query results show three different users: AMT, ALK, and JLR.

Who left the utility locks on processor 2?

```
SELECT CreateTime
,EventNum
,EventType (CHAR (12))
,UserName (CHAR (12))
,vproc
FROM DBC.Events_Configuration
WHERE vproc= ‘2’ ORDER BY 2 ;
```

```
 CreateTime       EventNum       EventType        UserName      Vproc

 --------------   -------------  --------------   -----------   --------
   14:06:22         1,153        Dump             AMT           2
   16:06:39         1,159        Dump             ALK           2
   18:12:09         1,164        Restore          JLR           2
```

Events_Media View

The Events_Media view provides information about ARC activities that used removable storage media. This information includes the volume serial numbers assigned to portable devices.

The following example requests the volume serial number of a restore tape. The query results show two restore operations, each with their own serial number and data set name.

```
SELECT EventNum
,EventType (CHAR (12))
,UserName (CHAR (12))
,VolSerialID
,DataSetName (CHAR (12))
FROM DBC.Events_Media
ORDER BY EventNum ;
```

```
 EventNum     EventType     UserName     VolSerialID      DataSetName

 ----------   -----------   ----------   --------------   ---------------
 79           Restore       PJ           MPC001           LDR.DMP1.JNL
 180          Restore       PJ           MPC002           RAN.DMP2.JNL
```

CHAPTER 10 Stopping and Restarting the System

This chapter provides information on stopping and restarting Teradata Database (otherwise referred to as Trusted Parallel Application or TPA in this chapter). If you are very unfamiliar with the utility commands discussed in this chapter, be aware that using these commands without first learning them or without guidance from Teradata support personnel can lead to unwanted results on the system.

Starting Teradata Database

You start the TPA differently depending on your OS platform.

Linux

At the command prompt:

1 Go to /etc/init.d/

2 Submit “tpa start”

MP-RAS

At the command prompt:

Submit "pcl -shell /etc/init.d/tpa start"

Windows

You can start Teradata Database using the Teradata Command Prompt or MultiTool.

Teradata Command Prompt

At the Teradata Command Prompt:

1 Click on Start > Programs > Teradata Database > Teradata Command Prompt

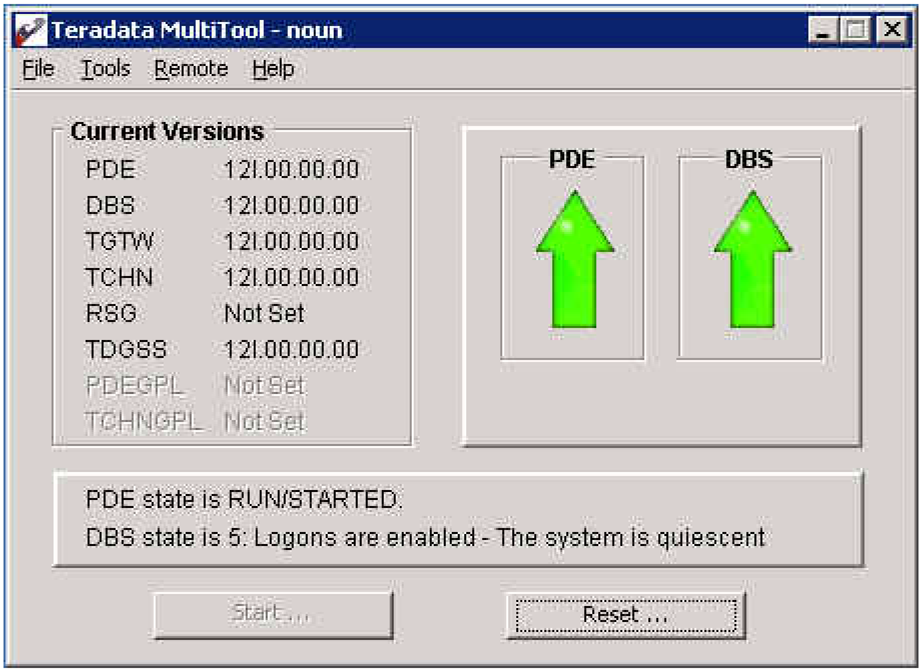
2 Type “net start recond” at the prompt and hit Enter. If successfully connected, you get the following message:

```
The Teradata Database Initiator service is starting..
The Teradata Database Initiator service was started
successfully.
```

MultiTool

1 Click on Start > Programs > Teradata Database > Teradata MultiTool

2 Click the Start button. MultiTool will ask if you want to start both the PDE and DBS or just the PDE only. To start Teradata Database, click both the PDE & DBS buttons.



Current Versions

PDE	121.00.00.00
DBS	121.00.00.00
TGTW	121.00.00.00
TCHN	121.00.00.00
RSG	Not Set
TDGSS	121.00.00.00
PDEGPL	Not Set
TCHNGPL	Not Set

PDE state is RUN/STARTED.
DBS state is 5: Logons are enabled - The system is quiescent

For more information on MultiTool, see *Graphical User Interfaces: Database Window and Teradata MultiTool*.

Stopping Teradata Database

To fix data corruption, update software, perform system maintenance, or make configuration changes, you must stop the Teradata Database system. To manually stop Teradata Database, do the following:

1 Log onto the system with administrator privileges.

2 Warn any active users that the Teradata Database is going to shut down. Wait until all active sessions have logged off and the system is quiescent.

3 Open a command prompt window.

Note: For Windows, you can use Teradata Command Prompt. Or you can alternatively use MultiTool to stop the TPA by clicking on the Reset button and selecting “Exit (Do not restart the TPA)”.

4 Regardless of operating system, you can enter the following command:

```
tpareset -x comment
```

The -x option stops Teradata without stopping the OS. You must enter a comment text string. This string appears in the Event Log.

For example:

```
tpareset -x stopping teradata for upgrade
```

Note: Use tpareset -x on both SMP and MPP systems. To stop the TPA on a single node of an MPP system running MP-RAS or Linux, use "/etc/init.d/tpa stop" and Teradata Database will stop on that single node while restarting Teradata Database on the other nodes.

5 When the system asks you if you are sure you want to bring the database down, type Y and hit Enter.

The system stops in an orderly sequence.

6 To verify that the system has stopped, submit the following command:

```
pdestate
```

- On Windows, the correct stopped status response is:

  ```
  PDE state is DOWN/HARDSTOP
  DBS state is DOWN
  ```

  MultiTool on Windows also displays whether or not the PDE and DBS are down. Two down arrows means the TPA has stopped.
- On MP-RAS, the stopped status response is:

  ```
  Parallel Database Extension state is NULL/STOPPED
  ```
- On Linux, the stopped status response is:

  ```
  PDE state is STOP/KILLTASKS.
  ```

If the node is a standby node, the pdestate command reports the node as `NULL/STANDBY`.

Restarts of the Teradata Database

When a restart occurs, Teradata Database performs the following:

- Generates an information message to each active system console
- Terminates all tasks and returns an informative message to the session
- Frees all associated resources. For example, spool cylinders are returned to the unused cylinder pool
- Initiates startup processing
- Performs transaction recovery processing

Note: If a restart occurs when Reconfiguration processing is in the middle of the Hash Map calculation phase (a rare occurrence), the operation might not be able to restart because of unexpected tables on the new AMPs. For instructions on how to recover, see "Reconfiguration Utility (reconfig)" in *Utilities*.

Restarts can also be either Cold or Coldwait as described in the following table.

Restart Type	Description
Cold (default)	Users can log into the system before transaction recovery processing is complete. The system does not wait for all AMPs to completely recover. Instead, the system places them in offline catchup and logons can be enabled before recovery is complete. The cold option can be set in Database Windows (DBW) or Vproc Manager.
Coldwait	Logons are blocked until the transaction recovery processing is complete. The system waits for down AMPs to fully recover and return to online state before allowing logons.

In addition, restarts can be:

- Automatic (cold restart) by the system
- Forced (cold or coldwait restart) by the database administrator

Automatic Restarts

An automatic restart is triggered by any of the following:

- A failure on a client (host) system.
- A failure within a TDP.
- One or more vprocs fail and automatically restart as a result of any the following:
  - A software error or vdisk failure.
  - Database, application, or BYNET software error in a single vproc.
  - Two physical disks fail in the same rank.
- Power surge after a power loss, if the “Restart After DAPowerFail” field in the xctl or ctl utility is set to ON. The default is ON.
- A hardware component fails.

In most cases, Teradata Database restart processing occurs automatically. Users logged on at the time of an automatic restart are informed of the occurrence. The type of information returned depends on whether the restart was caused by hardware, client, TDP, node, or software (vproc) failure.

The impact on your Teradata Database depends on the failed component, as explained in the following tables.

Component Hardware

<table>
<tr><th>IF the failed component is a …</th><th>THEN …</th></tr>
<tr><td>BYNET</td><td>processing resumes on the other BYNET. Performance may be impacted to some extent. (BYNET-initiated restarts never cause a dump.)</td></tr>
<tr><td>Disk</td><td>data may be lost. Tables with fallback continue to be 100% available. Tables without fallback are partially available.
Replace the disk drive and check table integrity (for detailed instructions, see “Moving Vprocs for Hardware Upgrades” on page 433).
If data needs to be recovered, see “Restoring After a Down AMP” on page 269 and “Startup and Recovery” on page 292.</td></tr>
<tr><td>Node</td><td>the impact on performance is minor if the system is configured with cliques. The other AMPs in the clique take over the responsibilities of the failed node and the system continues. To replace the physical node, see “Moving Vprocs for Hardware Upgrades” on page 433.
If your system is configured with a hot standby node, and a single node in a clique fails, the system remains available as the standby node becomes the new production node and the down node becomes the new hot standby node when it is repaired.
The vprocs of the failed node migrate to all the other available nodes including the newly joined hot standby node. You do not have to restart the system to re-introduce the failed node once it has been fixed. It remains as the new standby node.
You can use the PMON tool of the Teradata Manager application to report the status of the nodes. These are the following possible statuses:
<table>
<tr><th>Status</th><th>Description</th></tr>
<tr><td>U</td><td>Up, or part of the active configuration.</td></tr>
<tr><td>D</td><td>Down.</td></tr>
<tr><td>S</td><td>Standby node.</td></tr>
</table>
</td></tr>
</table>

If a disk array loses electrical power but the Teradata Database nodes do not, the following occurs:

1. The AWS notifies the TPA nodes of the disk array event.
2. Teradata Database maps the affected devices to the affiliated vprocs that were marked fatal.
3. Teradata Database issues a TPA reset and comes back up without the vprocs that were marked fatal.
4. Teradata Database notifies the AWS that it has finished handling the event.

To choose if the database automatically restarts after a disk array power failure, set the “Restart DAPowerFailure” parameter in the ctl or xctl utility. Set the parameter to OFF if you do not want the system to be automatically restarted after disk array power failure. For more information on how to set this flag, see “Control GDO Editor (ctl)” or xctl in *Utilities*.

Component Software

IF the failed component is...	THEN the impact is ...
an AMP	the system can continue servicing users with other AMPs, if a single AMP fails in one or more clusters. However, the performance level and response time slows down. If two or more AMPs fail in a single cluster, the system halts. All processing stops until you bring at least one of the AMPs back online. For more information, see "Restoring After a Down AMP" on page 269.
a PE	very little on system performance. The maximum session capacity is reduced, and logons may take longer.

Database Errors

For database errors, by default there is always a dump on all nodes. (For references on where to find more information on dumps, see "Chapter 15 Handling Teradata Crashdumps" on page 423.)

MP-RAS System Resets

The following table describes the types of resets can occur on a MP-RAS system.

Reset Type	Description
Database restart or TPA reset	All tasks for vprocs are killed and restarted across all nodes in the Teradata Database configuration. UNIX does not reset. Not all processes are killed. Many of the UNIX processes that create the TPA are cancelled and restarted. Most of the kernel daemons are not killed, and several other processes survive resets.
UNIX panic	The node resets, causing the database to reset also.

Forced Restarts

You can use the Database Windows Supervisor or Vproc Manager utility to force a restart on the system. However, you should only need to force a restart for the following reasons:

- To change the running version of Teradata Database.
- To register and globally propagate changes to certain fields in the DBS Control record, such as some user-tunable parameters.
- To force a PDE dump to be taken.
- To bring AMPs in "offline recovery" online.
- To register a changed default for a user-defined multinational character set or collation sequence.

Note: The default for saving dumps during a forced restart is always No. However, you can explicitly specify whether or not to save a dump.

Restarting On MP-RAS

On MP-RAS, the first step is to run the xctl utility to turn on the DBS Start flag. You must do this *first* in order for Teradata Database to recover correctly.

After you set the DBS Start flag on, you can force a restart manually by entering the command **restart tpa**. If possible, wait until no production processing or utility operations are active.

To restart Teradata Database on MP-RAS, use the procedure below.

Note: If more than one node is down in the same clique, before restarting you can tune the system to not start more vprocs than the remaining nodes can run. To do this, set the Minimum Node Action and Minimum Nodes Per Clique options (in the DBS Settings window) to the appropriate values.

1 First, use the xctl utility to turn the DBS Start flag on:

```
# /usr/ntos/bin/xctl -nw
> screen debug
> 0=on
> write
> quit
```

Warning: **You must perform this step *before* you start Teradata Database, otherwise the startup fails in a manner that makes it impossible to initialize it properly.**

For more details, see "Xctl (xctl)" in *Utilities*.

2 Start Teradata Database using one of the following commands:

IF PDE is...	THEN on any node...	Enter ...
operable	from DBW console	*restart tpa comment [cold/coldwait]*
	using Vproc Manager	`restart [cold/coldwait]`
	at an MP-RAS prompt	`tpareset [-option]` *comment*
not running	on each node	`/etc/init.d/tpa start`

The following options are available with the tpareset command:

Option	Description
-d	Specifies that a DBS dump be taken before doing the restart.
-f	Forces all TPA nodes to participate in the tpareset regardless of their state, without rebooting UNIX.
-l	Specifies delay interval in seconds that the system should wait for other nodes to join the TPA configuration.
-P	Requests the node to panic after the DBS dump is saved.

Option	Description
-Q	Requests tpareset to run in silent mode, that is, the system will not prompt the user for confirmation.
-x	Shuts down the Teradata Database on the entire system without shutting down UNIX.

3 Wait for the start to complete, then submit the “pdestate” command to verify that all nodes are in the TPA state.

Restarting On Windows

Restarts on Windows system include both automatic and manual restarts.

Automatic Restart

Teradata Database starts automatically on Windows after a reboot or startup. To view the status of a restart, select:

```
Start > Programs > Teradata Database > Database Window
```

Teradata Database is fully started when the Database Window status line displays:

```
Logons all enabled - The system is quiescent.
```

Or you can use MultiTool to view the status as well. Multitool will also report all logons enabled and a quiescent system (along with two green arrows) if the TPA is running.

Manual Restart

Use one of the following methods to manually start Teradata Database.

TO start Teradata Database from the...	THEN do the following...
MultiTool	1 Log onto the server as a user with administrator privileges. 2 Bring up the Teradata MultiTool window: 3 Start > Programs > Teradata Database > Teradata MultiTool 4 In the MultiTool window, click the Reset button. You can accept the default comment string or enter your own comments in the dialogue box. 5 Select the “Restart (restart the TPA)” radio button.
Teradata Command Prompt	1 Start > Programs > Teradata Database > Teradata Command Prompt 2 Submit the following command: *net start recond*

Changing Logon States and Restarting the System

You can control whether all users, no users, or only user DBC can log on to Teradata Database as described in the following table.

Changing the logon state while the system is...	Using the...	Means changes go in to effect...
up	Database Window Supvr (Supervisor Window)	immediately. No restart is required and the new logon state applies as soon as you submit the command.
down	ctl or xctl	after the DBS restarts.

The following table lists how logon states affect sessions before and after system restart.

IF the Logon State is...	Prior to Restart...	After Restart...
enable logons/ enable all logons	All users, including DBC, can log on to the system.	The system reconnects all external logon sessions that were previously connected. For example, logons through BTEQ, mainframe, or FastLoad automatically reconnect when the system is back up. The system permits new logons to connect.
disable logons/ disable all logons	No users, not even user DBC, can log on to the system. If the logon state is changed to "disable logons" when the system is still up, all users already logged on remain logged on. The only way to remove these sessions is to abort them. However, the system prevents new logons from connecting.	The system reconnects all external logon sessions that were previously connected. For example, logons through BTEQ, mainframe, or FastLoad automatically reconnect when the system is back up. The system does not permit any new logons to connect.
enable dbc logons	Only user DBC can log on to the system. If the logon state is changed to "enable dbc logons" when the system is still up, all users already logged on remain logged on. The only way to remove these sessions is to abort them. However, the system prevents new logons from connecting unless the logon is through user DBC.	The system reconnects all external logon sessions previously connected. For example, logons through BTEQ, mainframe, or FastLoad automatically reconnect when the system is back up. All other logon sessions remain aborted and the system only permits new user DBC logons or previously connected logons from user DBC.

Note: Regardless of logon state, upon restart, the system kills all internal logon sessions.

Use the tools listed in the following table to change the logon state.

Tool	Description
Screen debug commands of ctl or xctl utility	You can use screen debug in ctl or xctl to change the logon state when the DBS is down. Set the "Enable Logons" field to either: • ALL (default) • NONE • DBC For example, at the command line, you could type "2=DBC" to change the logon state to enable logons only for user DBC. When you exit, type Y to save the changes when the system prompts you, "Write Control GDO Changes?" **Note:** Changes to logon state only go in to effect after the system restarts.
Database Window Supervisor Commands	Use the Supervisor window if the DBS is up and running. Submit any of the following commands: • enable logons/enable all logons • enable dbc logons • disable all logons/disable logons **Note:** Changes to logon state go in to effect immediately.

Startup and Recovery

Restart processing takes place without manual intervention, even in the event of a hardware failure, as long as an operational configuration exists. An operational configuration consists of:

- Not more than one offline or down AMP in the same cluster.
- At least one available path to at least one copy of the data.

Usually, the configuration is operational. In some instances, such as after a power loss, your field support personnel may need to reconfigure it manually.

Database Startup and Recovery

The database startup and recovery phases involve the following phases. Teradata Database:

1. Starts the PDE/TPA layers.

   At TPA startup, the system logs an event for each node that is down and the BYNET service (BNS) reports:

   - Number of nodes online
   - Number of standby nodes used out of total nodes (only if standby nodes have been configured)
   - Number of nodes down (only if a down node exists)

- Total nodes (only if standby nodes are configured or the number of active standbys does not equal the number of down nodes)

2 On an MPP system, PDE recalculates the minimum number of nodes per clique required for normal operation.

   Note: If you manually changed the Minimum Nodes Per Clique option before the restart, and PDE determines that the set value is smaller than the automatically calculated value, PDE uses the automatically calculated value. For details, "Xctl (xctl)" in *Utilities*.

3 Starts the PEs and AMPs.

4 Attempts to bring up any down AMPs and reports AMP status.

5 Rebuilds any special system users and databases (see "The Initial Teradata Database" on page 36) and other system objects, as necessary.

6 Reads the Teradata Dynamic Workload Manager configuration settings and rebuilds the Teradata DWM rules cache if Teradata DWM was active at the time of the restart. Teradata DWM resumes query validation as soon as sessions are reconnected and logons are enabled. (For more on Teradata DWM, see "Managing Workloads with Teradata Dynamic Workload Manager" on page 318.)

7 Spawns dbscheck when "Logon Enabled" state is achieved (only if you have set dbscheck to run in daemon mode). Running as a background task on the tpa control node (lowest active node), dbscheck periodically samples DBS response time. If response exceeds the timeout interval, DBS issues the logevent; the event message includes the timeout value.

   To change the sleep and timeout intervals, run the dbscheck tool interactively from any node.

   To stop dbscheck, in Windows use <Ctrl>C or Task Manager. On MP-RAS, use the kill -pid command. (For instructions and examples, see "Resource Check Tools (dbschk, nodecheck, syscheck) " in *Utilities*.

8 Reads the syscheckrc file to register threshold levels used by syscheck. (For details, see "Resource Check Tools (dbschk, nodecheck, syscheck)" in *Utilities*)

9 Recalculates space consumption and updates the system tables.

10 Performs transaction recovery (see "Transaction Recovery" on page 294).

11 If Cylinder Read is enabled, Teradata Database instructs each AMP to map a view of its FSG cache into its virtual address space (see "Managing I/O with Cylinder Read" on page 309).

12 Reconnects all sessions. For network sessions, the Teradata Gateway:

   - Retrieves (from the PCLXSESSINFO parcel) the seed and hash of the password associated with each disconnected session.
   - Rehashes each password using the seed.
   - Compares the new hash value to the saved hash value.
   - Reconnects each session with a password hash match.

Transaction Recovery

Teradata Database automatically recovers all transactions that were running at the time of the restart. Transaction recovery involves determining whether transactions being processed by an online AMP when the database restarted should be committed or rolled back.

Note: Transient Journal (TJ) rollbacks on systems not using NUSIs have been optimized to block-at-a-time for user initiated aborts or system restarts. This provides significant improvements to performance of previous releases which used row-at-a-time rollback.

When an AMP completes work on a particular transaction, it places an End Transaction marker for that transaction in the TJ.

During transaction recovery, the AMPs examine their TJ for an End Transaction marker for every transaction in progress when the restart occurred. For a transaction to qualify as committed, the End Transaction marker must be found by every AMP involved. Otherwise, the transaction is rolled back.

Teradata Database automatically does the following for recovery:

1. Completes committed transactions and releases spool files
2. Recovers data from system recovery journals
3. Rolls back uncommitted transactions

   Note: For rows using identity column, if a roll back occurs, the assigned numbers may be lost and will not be re-generated for the same rows when the request is re-submitted.
4. When possible, the system reconnects sessions that were active at the time of restart

For more information, see Chapter 9: "Archiving, Restoring, and Recovering Data."

If a disk failed, replace the physical disk. See "Moving Vprocs for Hardware Upgrades" on page 433 for more information.

Using System Recovery Journals

The system recovery journals for completing transaction recovery include those listed in the following table.

Journal	Function	Location
Down AMP Recovery Journal, including: • Changed Row Journal (CJ) • Ordered System Change Journal (OSCJ)	• Contain information on updates to fallback tables that occurred while an AMP was down (offline). • Used for updating the fallback tables on the down AMP prior to bringing that vproc back online. **Note:** Non-fallback data on the down vproc cannot be updated while the vproc is down.	Remaining active AMPs in the cluster
Transient Journal (TJ)	• Determines whether in-progress transactions are committed • Completes committed transactions • Rolls back uncommitted transactions	Local to each AMP

Performing Online and Offline Catch-up

When Teradata Database undergoes a cold restart, the system must decide whether to perform the recovery process when an AMP is offline.

An AMP can usually catch up after being brought back online if the following are true:

- The Change Row Journal (CJ) count is less than 3000
- The Ordered System Change Journal (OSCJ) count is 0
- The Transient Journal (TJ) count is 0
- No HUT locks are present in the cluster

Otherwise, the AMP is kept current even though remaining offline. Every five minutes, a background task is activated that applies changes from the CJ and OSCJ to keep the offline AMP caught up until the situation is resolved.

Restarting the Database Window on MP-RAS

When startup is complete, on an MPP system you can restart the Database Window (DBW). On the control node (the lowest numbered node that is up), enter:

```
export DISPLAY = <IP address of PC>
#/usr/ntos/bin/xdbw
```

The outline for the DBW should appear on the screen. The window does not reconnect until restart is complete.

If an error window appears with the message:

```
Attempting to connect to CNS . . .
```

and remains in the middle of the DBW, it may be that Teradata Database restart is still in progress. Wait a few moments before trying again.

See *Graphical User Interfaces: Database Window and Teradata MultiTool* for more information.

Startup Event and Status Logs

Startup events and status are logged as follows:

- Status field to the DB Window
- DBC.Software_Event_Log (MP-RAS, Windows, Linux)
- */var/adm/streams* (MP-RAS)
- /tpi-data/nodecheck.tpacycle_*n* (MP-RAS)
- …\tdConfig\tpi-data\nodecheck.tpacycle_*n* (Windows)
- Windows Event Log (which can be viewed with Windows Event Viewer)

SECTION 4 Housekeeping and Maintaining the Database

CHAPTER 11 Recommended Housekeeping Tasks

The following chapter discusses some common tasks you should perform on a regular basis. Regularly cleaning up old logs, deleting old data, collecting statistics, and tuning queries are just some of the many things you can do to keep your database running at its best.

Note: Teradata recommendations are useful guidelines for maintaining the system. However, the frequency of how often to do these tasks, what should be the actual policies, and what are the appropriate procedures for your site depends upon your available resources and specific system requirements.

Routine Housekeeping Tasks

The following sections describe some general tasks that you should perform on a regular basis. Cleaning up old information, whether in tables or in collected statistics, helps your Teradata Database system access tables more quickly and maximize resources. Teradata recommends that you periodically archive the log data and then delete the information. However, be careful not to drop the tables.

Cleaning Out Frequently Updated Tables

The following logs and system tables are frequently updated and can consume valuable disk space. They are not purged automatically and should be monitored and maintained.

It is good practice to delete old information you no longer need. If you want to keep the data in the logs for historical purposes, disaster recovery, or trend analysis, you can always archive them first.

Use Teradata Manager or create your own BTEQ scripts to clean out the following:

- AMPUsage (if you are using ASE)
- AccLogTbl (if you are using Access logging)
- DBQL tables (if you are logging DQBL information)

  Note: You should schedule clean up activities during off-peak times. Otherwise, the delete process locks the DBQL table and if DBQL needs to flush a cache to the same table in order to continue logging queries, the whole system could experience a slow-down.
- HW_Event_Log
- InDoubtResLog
- LogOnOff

- ResUsage tables (if you are logging resusage information)

  Note: Teradata recommends you purge resusage data on a daily basis. You can use the Data Collection tab of the Administrate menu in Teradata Manager to specify the number of days you wish to keep the detail data. Then schedule Teradata Manager to run the Cleanup function. You can also manually submit SQL statements to delete old data from Resusage tables.

- SW_Event_Log

Note: When you periodically delete the entries in system tables, do ***not*** drop the table. For example, you can submit queries like the following:

```
DELETE FROM DBC.AMPUsage WHERE Accountname = '$M619';
DELETE FROM DBC.LogOnOff WHERE (DATE-Logdate) > 90;
```

Use the DeleteAccessLog and DeleteOldInDoubt views to delete logs more than 30 days old without specifying the date criteria. For example:

```
DELETE FROM DBC.DeleteAccessLog ALL;
```

Caution: Using the DELETE…WHERE statement causes a full table scan. Target tables will be locked for the duration of the operation. Therefore, schedule clean up activities during off-peak times.

Refreshing Stale Collected Statistics

Statistics are data demographics "hints" used by the Teradata Optimizer. There are many ways to generate a query plan for a given SQL statement, and collecting statistics ensures that the Optimizer will have the most accurate information to create the best access and join plans. Without collected statistics, the Optimizer assumes that any column or nonunique index is highly nonunique and will create join plans accordingly. You should collect statistics on any columns (referenced in a DML WHERE clause) or nonunique indexes that are fairly or highly unique so that the Optimizer can make proper use of them in join plans, and thus, enhance performance.

Statistics should be collected for:

- The primary index columns on small tables where the number of rows is less than 5 times the number of AMPs
- All columns defined as a NUPI in large tables
- All columns in your JOIN condition for all tables (that is, those columns in any DML WHERE clause).

Once collected, update statistics regularly to accurately reflect the data demographics of the underlying table. It is better to have no statistics than stale or otherwise incorrect statistics. A rule of thumb is to collect statistics when data has changed by more than 10 percent. (That would be 10 percent more rows inserted, 10 percent of the rows deleted, 10 percent of the rows changed, or some combination.) For a PPI table, update statistics every time more than 10 percent of a partition changes or is updated.

Note: Re-collecting statistics involves a full-table scan and may take a significant amount of time. Therefore, collect statistics off-hours for large tables. You can also execute “HELP

STATISTICS *tablename*" before and after recollecting statistics, to see the difference the recollect makes.

For frequently executed queries, requesting an EXPLAIN before and after recollecting statistics may show differences in join plans, spool row counts, and processing time estimates. Sample statistics take less time but may be less accurate.

Note: You can collect partition statistics on all partitioned tables using system derived column PARTITION. This helps the Optimizer do more accurate costing with partitioned tables which helps produce more optimal plans.

For more information on collecting statistics, see "COLLECT STATISTICS (Optimizer Form)" in *SQL Reference: Data Definition Statements*.

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 rights added to DBC.AccessRights. These entries should be reviewed and obsolete entries deleted as necessary. Whenever a logon is made to the database, a full table scan is done against this table. The smaller this table can be made, the better.

  This table can grow quite large and can greatly impact space limits negatively. To reduce table skewing, clean up redundant rows in DBC.Accessrights. Or convert to using roles and then remove the redundant rows.

Caution: Be careful when deleting rows from DBC.AccessRights. If you inadvertently delete rows for DBC on system tables, you may have to contact Teradata Support Center to remedy the problem.

- **Archive and recovery tables** - When associated removable media is expired and over-written, clean out the following:

Table	Purpose
DBC.RCConfiguration	Archive/Recovery config
DBC.RCMedia	VolSerial for Archive/Recovery
DBC.RCEvent	Archive/Recovery events

- **Accumulated values in Data Dictionary tables** - Resetting values can help you do better trend analysis, resource usage analysis, and keep cleaner historical records. The following fields should be reset when necessary:

Fields	Purpose
Fields such as MaxPermSpace, MaxSpoolSpace, and CurrentPermSpace in the DBC.DatabaseSpace table.	Contains database and tablespace accounting information. To reset the values, execute the macro: `EXEC DBC.ClearPeakDisk;` 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.

CPUTime and DiskIO fields of the DBC.Acctg view (DBC.AMPUsage table).	Holds resource usage data by Acct/User. To reset the values, submit the following:
	`UPDATE DBC.AMPUsage`
	`SET CPUTime = 0,`
	`DiskIO = 0`
	`ALL;`

AccessCount and LastAccessTimeStamp fields of the following views: • DBC.Columns • DBC.Databases • DBC.Indices • DBC.Tables • DBC.Users	Tracks the number of times and the last time a particular object was accessed. To reset, use one of the following DIPVIEW macros: • ClearAllDatabaseUseCount • ClearDatabaseUseCount • ClearTVMUseCount

Tasks for Housekeeping Disk and Table Space

The following table provides suggestions for how often you should run certain utilities or miscellaneous tools to manage space and improve the performance of your system.

Task	Recommended Frequency of Use	Comments
Run the SHOWSPACE command of the Ferret utility	Daily basis	You can use SHOWSPACE of the Ferret utility to check disk space utilization and the percent of space free per cylinder. If it is low, try deleting obsolete tables or logs. Or, you can run PACKDISK. Freeing space on the system as often as you can helps reserve space for spool or for future data and helps improve performance.
Run the PACKDISK command of the Ferret utility	Weekly basis during off-peak times	Reclaim cylinders with spare datablocks. (Run the SHOWFSP command to get an estimate on which tables would best benefit from packing.) Running PACKDISK reverses the effect of cylinder splits and packs the cylinders full of data, leaving only the percentage of space indicated by FreeSpacePercent empty.

Task	Recommended Frequency of Use	Comments
Run CheckTable utility	Monthly basis	CheckTable is useful for finding problematic tables and checks for inconsistencies among internal data structures such as table headers, row identifiers, and secondary indexes. **Note:** Although CheckTable identifies and isolates data inconsistencies and corruption, it cannot repair inconsistencies or data corruption. The following options are available: • ERROR ONLY option - CheckTable can report only the failed tables with warnings and errors (rather than on all the tables it checked). • CONCURRENT MODE option - To run CheckTable on an active production system (that is, non-quiescent system). • PARALELL MODE - Specify the number of tables checked in parallel. For more information, see *Utilities*.
Run Scandisk command from the Filer utility	Monthly basis	Best practices include performing online during non-peak times. Previously halted SCANDISK operations can be restarted using a perl script. For more information, see "Filer Utility (filer)" in *Utilities*. The TABLE option allows you to specify starting table and ending table options. **Note:** 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.

Updating Software

Updating your software to the most current patch levels can sometimes help eliminate problems. 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 Teradata Database, as well as other software on your system, as current as reasonably possible.

It is good practice to monitor Tech Alerts and known fixes. Performance problems may actually be symptoms of other problems.

To obtain an EFix, you will need your Teradata support personnel to open an Etape call log. This begins the process for Teradata to develop the required package. Your Teradata support personnel will open a change control for the application of the patch when it is available. This change control must be approved by Teradata Support Center (TSC). In order to download the package, you must have the designated Etape call-log number and approved change control number.

When 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.
- The gateway version must match the PDE version.
- It is okay to have a higher version of PDE than DBS. However, you cannot have a higher version of DBS than PDE.

For more information, contact your TSC personnel. Or, if you have a service account, check the service website. For more information on performance issues, see *Performance Management*.

Housekeeping Queries

It is good practice to regularly tune and check your queries to make sure they are running optimally. This section introduces some tools you can use to take best advantage of the query analysis capabilities of Teradata Database, including:

- "Target Level Emulation (TLE)" to replicate your production configuration in a safe test environment. Test and tune queries on this test environment before implementing changes on your production system.
- "Query Capture Facility (QCF)" for index analysis, using an SQL interface, to capture data demographics, collect statistics, and implement results.
- "Teradata Index Wizard" for analysis and recommendations, using data captured via QCF or DBQL capabilities.
- "Teradata Visual Explain Utility" to compare results from a query run at different times, on different releases, or with different syntax.
- "Building Baseline Transaction Profiles" using results derived from Teradata Performance Monitor and Resource Usage reports.

There are other tools that can help you optimize your queries such as Teradata Statistics Wizard, Teradata System Emulation Tool, and various tools in Teradata Manager not detailed in this book.

Target Level Emulation (TLE)

Target Level Emulation (TLE) simulates a production system environment on a test system and uses the same collection of cost variables used by the Optimizer to come up with the best query plan.

Note: While this feature produces a query plan for the emulated target system, it does *not* emulate the performance of that system.

TLE is also used by the TSC to emulate your production system for the purpose of query execution plan analysis. Query plans are generated on the test system as if the queries were submitted on the production system. TLE achieves this by emulating the cost parameters and random AMP samples of your production system on the test system. The following tools can then be used on the test system:

- Query Capture Facility (QFC) to capture the query execution plan.
- Teradata System Emulation Tool (Teradata SET) to:
  - Export the information needed for emulation from your production system.
  - Import that information on the test system.
- Teradata Visual Explain utility to capture query plans on the emulated test system and fine tune your queries before running them on the production system

These features allow you to develop and analyze query plans without impacting your production system.

Caution: The TSC should run TLE on a test system; do not enable it on a production system.

For more information on TLE, see *SQL Reference: Statement and Transaction Processing*. For more information on Teradata SET, see *Teradata System Emulation Tool User Guide*.

Query Capture Facility (QCF)

The Query Capture Facility (QCF) allows you to capture the steps of the query execution plans. The query text and plans are stored in special relational tables that you create in a user-defined Query Capture Database (QCD). The captured data is used to gather data demographics and analyze indexes for the tables referenced in the queries.

The source of the captured data is produced by the Teradata Database Optimizer, which outputs the text of SQL EXPLAIN detailing the final stage of optimization. (Although the current implementation of QCF includes the explain text, it does not represent all the information reported by EXPLAIN in relational form.)

The captured information is your workload, which becomes source input to:

- Teradata Database Statistics Wizard utility.
- The Teradata VE utility, which presents a graphical view of the flow of data during query execution. It also compares different EXPLAINs, either of the same query run on different Teradata Database configurations, or of semantically identical but syntactically different queries.
- The Teradata Index Wizard, which recommends index definitions that should improve the overall efficiency of the workload. Recommendations can include adding or deleting SIs to or from an existing design (see “Teradata Index Wizard” on page 306).

The query analysis statements are usually invoked by the Teradata Index Wizard or other client-based Teradata Analyst tools, but you can also submit them from a BTEQ script or interactive session. However, QCF uses Optimizer-based collect statistics recommendations.

For more information and a full discussion on how to create and use QCF, see “Query Capture Facility” in *SQL Reference: Statement and Transaction Processing*.

Teradata Index Wizard

The Teradata Index Wizard uses the contents of the tables in QCD to analyze specific workloads and suggest new indexes to improve query performance. The recommendations are for secondary indexes, join indexes (single-table as well as global), and partitioned primary indexes. Teradata Index Wizard may recommend the deletion of existing indexes as well as the addition of new indexes. Use the recommendations to evaluate potential performance improvements and modify your database accordingly.

Teradata Index Wizard offers the following benefits:

- Simulates candidate SIs, JIs, and PPIs without incurring the cost of creation.
- Validates and implements index recommendations.
- Provides automatic "what-if" analysis of user-specified index candidates.
- Interfaces with the Teradata System Emulation Tool to allow workload analysis on test systems as if the workload had been analyzed on the production system.
- Interfaces with the Teradata VE utility to compare query plans in the workloads.

Teradata Index Wizard is one of the software tools that make up the Teradata Analyst Pack. For more information, see *Teradata Index Wizard User Guide*.

For information on how to prepare the database for analysis by the Teradata Index Wizard, see "Database Foundations for the Teradata Index Wizard" in *SQL Reference: Statement and Transaction Processing*.

Teradata Visual Explain Utility

Teradata Visual Explain (Teradata VE) client-based utility is a powerful interface for application performance analysis and comparison. It generates a graphical view of the query processing sequence for a given SQL query. This can help you better understand why the Optimizer chooses a particular plan. You can use Teradata VE to:

- Tune performance by viewing how skewed data or missing collected statistics affects a query.
- Capture query plans on an emulated test system and fine tune your definitions before moving to a production system.
- Compare the same query run on differing releases or operating systems.
- Compare queries that are semantically the same but syntactically different.

The results can help you understand changes to the Teradata Database schema, physical design, and statistics. For more information, see *Teradata Visual Explain User Guide*.

You can also use Teradata VE with the Teradata Index Wizard. Teradata VE, along with Teradata Index Wizard, is one of the tools in the Teradata Analyst Pack.

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)
- 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
- DBQL (see Chapter 13: "Tracking Processing Behavior with the Database Query Log (DBQL)")
- Teradata VE utility (see "Teradata Visual Explain Utility" on page 306)

For more information on building baseline transaction profiles, see *Performance Management*.

For More Information

For information on other query capture, resource management, and resource monitoring tools, see:

- Chapter 12: "Tools for Managing Resources"
- Chapter 13: "Tracking Processing Behavior with the Database Query Log (DBQL)"
- *Performance Management*
- *Teradata Manager User Guide*

CHAPTER 12 Tools for Managing Resources

This chapter discusses tools and features useful for managing resources in the database so that you can minimize the occurrence of impeded performance, maximize throughput, and manage the consumption of resources.

There are several tools and features you can use.

IF you want to ...	THEN use ...
manage and maximize I/O efficiency	the cylinder read feature. See "Managing I/O with Cylinder Read" on page 309.
manage system resources priorities of jobs	Priority Scheduler. See "Managing Resources with Priority Scheduler" on page 312.
dynamically manage resource utilization, throughput, and workloads	Teradata Dynamic Workload Manager (Teradata DWM) See "Managing Workloads with Teradata Dynamic Workload Manager" on page 318.
customize the delegation of resources among the various workload processes	
find unused objects and delete them from the database	the instructions in "Tracking System Object Usage" on page 328.
determine which objects the system heavily accesses	
see the number of available AMP worker tasks (AWTs) or the number of messages waiting on a particular AMP	the ampload utility or the awtmon utility. For a description of the differences between the two, see "Differences Between ampload and awtmon Utilities" on page 395.

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 Teradata Database file system. Data blocks are stored in physical disk sectors or segments, which are grouped in cylinders.

Teradata Database employs Cylinder Read to enable 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. Cylinder Read is disabled if the Teradata File Segment (FSG) memory per AMP is below 36MB.

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.

When Cylinder Read is enabled (the default), it is invoked implicitly based on memory conditions and the query statement itself. The processing sequence is described in *Performance Management*.

The Teradata File Segment (FSG) Functions

The Teradata FSG subsystem is used by the AMPs for reading, writing, and caching data to and from the disks.

FSG maintains a cache of disk segments that are allocated to data blocks for:

- Permanent data rows (including secondary index and fallback rows)
- Spool space
- Cylinder Indexes (CIs), for managing permanent and spool data blocks
- Transient Journal (TJ) rows
- Recovery journal rows
- Synchronized scan (sync scan) data

Memory Slots in FSG Cache

An FSG segment is the basic unit of memory buffer provided by the PDE for the Teradata Database File System to manage and access data. When a task requires an FSG segment, the corresponding data is mapped into the FSG virtual address space.

With Cylinder Read, the FSG cache can be viewed as consisting of two regions, the Cylinder Pool and the Individual Segment. The Cylinder Pool is cut into cylinder-sized memory slots. The size of each slot is 1936KB (equal to 484 pages of memory).

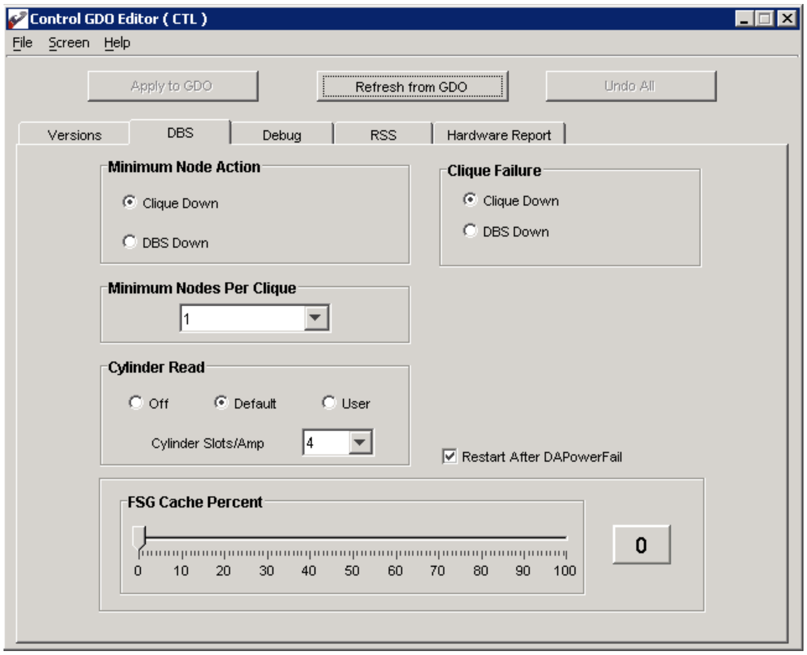
Changing the Cylinder Read Defaults

When the Teradata Database is installed, Cylinder Read is enabled by default.

Note: If you notice a performance hit from Cylinder Read, try increasing the default number of cylinder slots per AMP. See “Control GDO Editor (ctl)” and “Xctl (xctl)” in *Utilities* for more information on the Cylinder Read settings in DBS screen.

The feature is disabled automatically if FSG memory is below 36 MB per AMP. You can manually disable Cylinder Read, re-enable Cylinder Read, or change the number of slots per AMP using xctl (MP-RAS), or ctl utility (Microsoft Windows and Linux).

IF Cylinder Read is set to ...	THEN ...
DEFAULT	the value for Cylinder Slots/AMP is calculated automatically. If you set the slider to a value, the setting is ignored.
USER	you can set the Cylinder Slots/AMP value yourself. However, based on FSG cache size, in rare cases FSG may have to change the number of slots per AMP. (Teradata recommends that as a general rule, the default setting should provide the best performance.)



Minimum Node Action: ● Clique Down ○ DBS Down

Clique Failure: ● Clique Down ○ DBS Down

Minimum Nodes Per Clique: 1

Cylinder Read: ○ Off ● Default ○ User

Cylinder Slots/Amp: 4

☑ Restart After DAPowerFail

FSG Cache Percent: 0

For instructions on setting Cylinder Read parameters in DBS Screen, see “Control GDO Editor (ctl)” and “Xctl (xctl)” in *Utilities*. For an explanation of how to use Cylinder Read for best results, see *Performance Management*.

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 xctl utility (on MP-RAS) or ctl utility (on Windows and Linux). To determine size, see “Using, Adjusting, and Monitoring Memory” in *Performance Management*.

Tracking Cylinder Read Resource Usage

There are a number of fields in the DBC.ResUsageSvpr table that track Cylinder Read behavior if you enable ResUsage logging for the DBC.ResUsageSvpr table. For more information on these cylinder read fields in Resource Usage, see *Resource Usage Macros and Tables*. For information on how to best apply Cylinder Read, see *Performance Management*.

Managing Resources with Priority Scheduler

Priority Scheduler is a resource management tool that controls system resource allocations based on a blueprint that you construct to satisfy your site-specific requirements.

Every session logs on to Teradata Database with an execution priority established by an assigned or default Performance Group (PG). The PG defines how much CPU resource and processing time can be allocated to each query, thus effectively controlling query completion time.

Prioritization is active in all Teradata Database systems at all times. Teradata Database itself assigns different priorities to its own internal work.

Note: If Teradata Dynamic Workload Manager (Teradata DWM) Category 3 is active, prioritization is based on the resource scheme that was defined through Teradata DWM. Modification of scheduler parameters through the Priority Scheduler Administrator (PSA) or the command line interface is not allowed if Teradata DWM Category 3 is active.

With Priority Scheduler, you can:

- Balance resource usage across different applications and utilities
- Authorize users for access to prioritized levels of service based on the PG, carried in the user account string
- Dynamically alter the PG of a user or (with profiles) user group
- Regulate access to AMP worker tasks (AWTs)
- Dynamically modify parameters that define your scheduling strategy, plus:
  - Record these parameters as profiles
  - Automatically change the profiles at scheduled times
- Set CPU usage limits at a variety of levels

The parameters available for defining your scheduling strategy include:

- A prioritized weighting system
- Methods for dynamically adjusting your strategy based on resource use or calendar schedule.

These capabilities allow you to control your workload flow.

Depending on the functions you want to perform, the utilities, tools, and information sources for implementing Priority Scheduler may include those listed in the following table.

IF you want to ...	THEN use ...	AND refer to the following...
establish, modify, or delete Performance Groups, Resource Partitions, Allocation Groups, and other scheduler parameters	one of the following: • schmon command-line utility • Priority Scheduler Administrator (PSA) facility of Teradata Manager	• In *Utilities*: • “Priority Scheduler” • “schmon Utility” (Windows and Linux) • “Changing the Performance Group of a Running Job” on page 314 • “Priority Scheduler Administrator (PSA)” on page 313 • “Priority Scheduler Administrator (PSA)” in *Teradata Manager User Guide*
• easily define the Priority Scheduler Definition Set (formerly known as the profile parameters) • generate schmon scripts to implement the profiles	the PSA facility of Teradata Manager	
assign users to PGs	CREATE/MODIFY USER or CREATE MODIFY PROFILE statements	• “Implementing Profiles” on page 141 • “Accounts and Performance Group Names” on page 146 • *SQL Reference: Data Definition Statements*
control query processing based on account or PG or workload definition	• schmon or PSA to dynamically raise or lower the priority of an active session • Teradata Dynamic Workload Manager to restrict unqualifying queries and set up rules of submission or workload definitions for qualifying queries.	• “Viewing Account Strings and Session Statistics” on page 147 • “Managing Workloads with Teradata Dynamic Workload Manager” on page 318
collect performance data for near-real-time monitoring of your scheduling strategy	• Performance Monitor API MONITOR requests • Teradata Performance Monitor	• *Performance Management* • *Teradata Manager User Guide*
collect processing statistics for post-session comparison and analysis	• Priority Scheduler monitor output • DBQL feature to log query processing behavior • On channel-attached clients, the TDPTMON routine to track transaction and response times from and to the TDP • ResUsage macros to obtain reports of disk and CPU usage	• “Applying the Logged Data” on page 371 • “RSSmon Utility” in *Utilities* (MP-RAS only) • *Resource Usage Macros and Tables*
get resource usage statistics by PG	the DBC.ResUsageSPS table	*Resource Usage Macros and Tables*

Priority Scheduler Administrator (PSA)

PSA is a utility running under Teradata Manager that provides the ability to:

- Easily define Priority Definition Set (formerly known as Priority Scheduler profiles)
- Generate schmon scripts to implement those profiles

PSA also provides tools to monitor and control the Priority Scheduler environment.

A Priority Definition Set is the collection of Resource Partitions, Performance Groups, Performance Periods, Allocation Groups, and other scheduler definitions that control how the Priority Scheduler manages and schedules session execution.

The features provided by Priority Scheduler Administrator include:

- An easy-to-use GUI for standard use of the schmon utility
- Visualization of:
    - Assigned Priority Scheduler weights
    - CPU consumption, by Allocation Groups and Resource Partitions
- A Priority Definition Set configuration, with the ability to set up and save different profiles for different performance windows
- Priority Definition Set management and scheduling
- Enhanced ability to monitor Priority Scheduler performance
- Historical records of Priority Scheduler monitor output that can be charted and reported
- Pass scheduling data to the Session Applet of Teradata Manager Service

For details and instructions, see *Teradata Manager User Guide*.

Changing the Performance Group of a Running Job

You can manage queries and resources by changing the performance group of the account under which a session is running. Change these parameters at the session or query level, depending on the session being changed and the authorization of the user submitting the change.

A dynamic account change is accepted only if the specified account has been already defined for that user in a CREATE/MODIFY USER/PROFILE statement.

DBC.Accounts is searched first for a corresponding profile/account pair, then a user/account pair, to verify that the user can run under that account.

The following table describes rules and facilities for changing the performance group or account of the requests of an active session.

You can change ...	At this level ...	With this tool ...	Using ...

your own account or priority, if the account has been defined for you	• Session
	• Request
		Teradata SQL statement	• Static statements such as CREATE or MODIFY USER or PROFILE
			• Dynamic statements (interactive or embedded) such as:
			`SET SESSION ACCOUNT = 'priority/account' FOR SESSION sessionID`
			(Affects the remainder of your session until you submit another change.)
			`SET SESSION ACCOUNT = ‘priority/account’ FOR REQUEST requestID`
			(Affects only the subsequent request.)

You can change ...	At this level ...	With this tool ...	Using ...
the account of another user if you have at least these administrator privileges: • CREATE USER • ABORTSESSION	Session	Static: Teradata SQL	Static: MODIFY USER
		Dynamic: Performance Monitor (Teradata Manager)	Dynamic: "Modify a user account/priority string" operation. This operation lets you change to any level, regardless of profile definitions.
		PM/API **Note:** PM/API keeps group names in volatile memory, not in DBC.Sessiontbl, so a restart could void a SET SESSION ACCOUNT request. To be sure your change completes, use SET CRASH to specify NOWAIT_TELL, or verify the result.	SET SESSION ACCOUNT sessionID [hostID] 'priority/account' [Y/N] where Y or N defines how you want the change to be applied, as follows: • Specify Y and the change applies to all current and future requests of this session. If no steps are executing, affects the next request and DBC.SessionInfo shows the new priority or account for this session ID. • Specify N and only to the current request for the specified session. If no steps are executing, the next request for the specified session has the old account/priority.
	Request	PM/API	SET SESSION ACCOUNT requestID sessionID [hostID] 'priority/account' [Y/N]

Changing the Priority of a Query

If you are assigned to multiple account strings, each with a different performance group, you can change your session to one of the other accounts you are assigned to in DBC.Accounts. This means that you can change *only* to an account for which you are authorized and can only run queries with the priority associated with that account.

In most cases, you should not raise the priority of a query unless it is business critical. Typically, you lower the priority of a query to give other users access to more resources. For instance, you can lower the priority of a long-running job to free more resources for other jobs.

There are two ways to change the priority of a query:

- Change the priority of your own request or session. You can change the priority of the performance group of the next query you run or change all your queries for the remainder of the session. You can specify any level for which you are authorized.
- Change the priority of queries that belong to another person. You must have administrative privileges (at least CREATE USER, CREATE PROFILE, and SESSION

ABORT) to manipulate the priority of another user. Using the PM/API and Teradata Manager tools, you can change the priority of any session or query to any higher or lower level, including R, regardless of user or profile definitions.

Use the following table for more information on how to change the priority of a query.

IF you want to change the priority of...	THEN use ...	For further information, see ...
your next query	• SET SESSION ACCOUNT... FOR REQUEST • PM/API SET SESSION ACCOUNT	• "SET SESSION" in *SQL Reference: Data Definition Statements* • *Teradata Preprocessor2 for Embedded SQL Programmer Guide* • "SET SESSION ACCOUNT" in *Workload Management API: PM/API and Open API*
all your queries for the remainder of the current session	• SET SESSION ACCOUNT... FOR SESSION • PM/API SET SESSION ACCOUNT	• "SET SESSION ACCOUNT" in *Workload Management API: PM/API and Open API* • *Teradata Manager User Guide* • Teradata Manager online help for Performance Monitor
the running query or active session of another user	• Performance Monitor, "Modify a user priority/account string" operation • PM/API SET SESSION ACCOUNT	

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 or transaction 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 when you enable query logging. See Chapter 13: "Tracking Processing Behavior with the Database Query Log (DBQL)."

  For example:

```
Sel Username (Format 'x(10)'), queryband(Format 'x(40)'), AMPCPUTime from qrylog
where ampcputime > .154;

 *** 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 Dynamic Workload Manager and associate query bands with Teradata DWM filter rules or define them as workload definition attributes. See *Teradata Dynamic Workload Manager User Guide*.

You can also customize your own method of obtaining query banding information through using open API functions and stored procedures. For more information, see *Workload Management API: Open and PM/API*.

Defining Name-Value Pairs

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 syntax and usage rules, see “SET QUERY_BAND” in *SQL Reference: Data Definition Statements*.

Managing Work with Teradata Active System Management

Teradata Active System Management (Teradata ASM) is a set of products, including system tables and logs, that interact with each other and a common data source. It facilitates automation in the following four key areas of system management:

- Workload management
- Performance tuning
- Capacity planning
- Performance monitoring

With careful planning, Teradata ASM can improve and optimize your workload management and performance. It can also improve response times and ensure more consistent response times for critical work. This can reduce the effort required by DBAs.

Some of the products that make up the Teradata ASM solution include Teradata Manager, Teradata Database Workload Manager, and Teradata Workload Analyzer.

For more information on how to set up rules and workload definitions to help Teradata ASM automate system management, see the following section “Managing Workloads with Teradata Dynamic Workload Manager” and *Teradata Dynamic Workload Manager User Guide*.

Managing Workloads with Teradata Dynamic Workload Manager

Automatically manage how and when queries should run using Teradata Dynamic Workload Manager (Teradata DWM)[1]. Teradata DWM allows you to manage workloads through filtering, throttling, and prioritizing queries against rules you define, so that you can adjust behaviors under different system conditions and operational environments.

Teradata DWM can help avoid unusual situations that would otherwise require manual intervention. For example, you can employ certain rules and actions to take place when specific events occur or make the system dynamically adjust behaviors in the case of a specific event. You can create rules that allow high priority work to run while you throttle lower priority work to be delayed until the system can optimally manage it.

Teradata DWM can help you:

- Filter out queries based on restrictions such as resource limits or statement type. Queries that do not pass the rules are rejected.
- Control the flow of queries or utility sessions for concurrency and delay them if necessary.
- Classify queries into "workload definitions" where you specify classification criteria along with behaviors such as throttle limits, exceptions, and priority settings.
- Create global exceptions.
- Manage system conditions (SysCons) or operational environments (OpEnvs). (SysCons and OpEnvs are defined in the section "Managing the System Through a State Matrix" on page 322.) Define combinations of events that trigger a SysCon. For example, you can define a normal, a degraded, and a problem SysCon. The combination of current OpEnv and a just-triggered SysCon can result in a state change that changes workload management behaviors.

Teradata DWM Rule Categories

There are three categories of Teradata DWM rules. These rules are stored in the TDWM database. To make a rule effective, enable the rule and activate its corresponding Rule Sets category. The types of rules you can set are described in the following table.

Category Type	Description
Filter Rules	Restrict access to the system based on: • **Object Access**: Limit access to or from specific database objects. For example, set a rule to limit access to a certain table or from a certain user. • **Query Resource**: Permit or deny access to database resource based on things such as estimated row counts, processing time, joins, whether it causes a full table scan, and so on.

[1] You can access Teradata DWM directly or from Teradata Manager depending on how your system was installed and configured. For more information, see *Teradata Dynamic Workload Manager User Guide*.

Category Type	Description
Filter Rules (continued)	Session and query requests that reference restricted objects are immediately rejected. For example, you can create a rule to ban UserA and UserB from issuing any requests. Or you could create a rule that prevents any DML by anyone on TableC. SQL queries from any supported Teradata Database interface (including, but not limited to, BTEQ, CLIv2, ODBC, JDBC) are validated against these user-defined rules stored in tables in the DBC.TDWM database. Queries that are rejected are logged in the TDWMExceptionLog table and DBQLogTbl if DBQL is enabled for that user. You can make filter rules global or make the rules only just warn but not actually limit access. **Note:** Unless otherwise specified, Teradata DWM checks every query of every session. However, users "DBC" and "TDWM" are always bypassed. You can set up other users to also bypass TDWM checking.
Throttle Rules	Throttle incoming work based on: • **Object Throttle**: Manage how many sessions and/or queries can concurrently run for specific database objects. • **Utilities Throttle**: Define how many utilities can run, either individually per type of utility or collectively for total number of utility jobs for the entire system. You can control if queries or sessions are rejected or delayed. **Note:** Values set in throttle rules for load utility concurrency override the DBS Control MaxLoadTasks value. (Setting a rate through Teradata DWM rather than the DBS Control utility allows you to avoid having to change the DBS Control setting.) You can specify limits for a mix of utilities. For example, you can define rules to allow only up to 3 FastLoad jobs, 2 ARC jobs, and 6 load jobs in total for the entire system. If there are any changes to the rules with respect to throttle values, the running queries are not subject to any new delay or abort directives.
Workload Definitions	Workload rules provide more complex management of queries. Within each workload, set criteria for: • Which queries to include (classification criteria) • Throttle limit • Priority Scheduler mappings (Workload Definition to Allocation Group) • Run-time exceptions • Service Level Goals (SLGs) for reporting and analysis The DBS classifies a query into a Workload Definition (WD). It takes the query attributes and puts the query into the correct WD based on the classification criteria. Classification criteria include things like estimated processing time, estimated row count, statement type, query bands, user id, account name, application name and so on. Each WD is associated with a Performance Group (Priority Scheduler groups)[a]. Throttle values and exception handling directives can also be placed on WDs. Exception handling directives allow a running query to be monitored and acted upon if conditions are met. You can define up to 36 different workload class definitions. Teradata provides four standard definitions (R, H, M, & L) plus a Default WD for query requests that do not fall into any of the other WDs. You can also create global exceptions that apply to multiple or all WDs or you can create multiple local exceptions that apply to only one WD. Exception directives allow a running query to be monitored and acted upon if conditions are met. Exception criteria include things like CPU consumption thresholds, CPU to I/O ratio, skew, and so on.

[a] The Priority Scheduler Administrator is still available to control Priority Scheduler settings if Workload rules are not activated.

You only need to specify those objects that you want restricted by a rule or workload. If a user or table, for example, is not associated with any Teradata DWM rule, then Teradata Dynamic Workload Manager does not restrict that object. However, if a user not affected by a rule tries to access a table that is restricted by a rule, that table restriction will still apply to the request issued by the user.

Note: By applying a rule to a database, the rule also affects all of the tables, views, macros, and stored procedures in that database. (Also, note that the underlying objects within views and macros will be resolved to the originating object.)

The TDWM GDO stores the current or activated set of rules. The EVENTS GDO contains those events with durations (that is, user events and SysCons), insuring their survival in case of a system reset.

The following DBC tables log Teradata DWM information:

- TDWMEventLog stores information on occurrences that affect Teradata DWM such as rule changes.
- TDWMExceptionLog contains information on exceptions.
- TDWMSummaryLog stores WD summary data.
- TDWMEventHistory logs the history of all event and state changes.
- SystemQTbl is available as a means for applications to be notified of events.

For more information, see *Teradata Dynamic Workload Manager User Guide*.

Rule Sets

A rule set is a complete set of the following rule types:

- global settings
- filters and throttle rules
- workload definitions
- Priority Scheduler settings

Teradata DWM enforces the rules for one rule set at a time. Each individual rule defined within a rule set consists of fixed attributes that never change (regardless of the state) and "working value" attributes that can be different for each state. Each unique state matrix corresponds to a set of working values for the rules defined in the rule set.

For example, say there is a throttle rule on user Joe. A state change can cause the throttle value to change from 12 to 8. While the name attribute "Joe" is fixed and cannot change, the throttle value is variable and part of the "working value" set that is different in each state defined. This means user Joe is allowed to run a different number of queries based on the state of the system.

The working values of the rule set, or working value set, includes all the limits for all the rules that apply when a state is active.

Examples of Teradata DWM Rules

The supported Teradata DWM rules allow you to set the limits in the following table.

Note: To make a rule effective, first enable the rule and activate its corresponding Rule Sets category.

IF you want to …	THEN define…	AND results will be based on …
control access to database objects	an access restriction and then link the database objects to be restricted.	states when the restriction is in effect.
control query processing time	a maximum processing time limit using a query resource restriction and then link the appropriate database objects.	• States when the restriction is in effect • Elapsed time estimated by the Optimizer
control response rows	a maximum row limit using a query resource rule and then link the database objects to be restricted.	• States when the restriction is in effect • Answer set size estimated by the Optimizer
control how many load utilities run simultaneously at any given time	rules for load utility throttle with desired limits.	• States when the rule is to be in effect • Current number of active load utilities
limit types of joins or scans	rules on types of joins or full table scans using a query resource restriction and link the database objects to be restricted.	• States when the restriction is in effect • Join and scan processing as determined by the optimizer
limit or delay query processing	a workload rule and then link the user, account, or performance group whose sessions or queries are to be limited or delayed.	one or a combination of the following: • States when the rule is to be in effect • Number of sessions running when the query is received
give all tactical queries high priority and strategic queries a lower priority	WDs with criteria that uniquely identify a tactical query from a strategic query using classification criteria such as query band, user id, estimated processing time, and so on.	query attributes.
give console utilities a priority	assign them to an appropriate WD.	
uniquely manage many different jobs from a single application using session pools rather than managing the entire application in common under one workload with one priority	different query bands for each type of job associated with multiple WDs	the requests classified by different query bands will run under different priorities.

For more examples, see *Teradata Dynamic Workload Manager User Guide*.

Setting Up Exceptions

Define exception criteria for workload definitions and if they are met, Teradata DWM logs the exception and performs the exception action defined. Actions triggered for exceptions include aborting requests, raising an alert or running a program through Teradata Manager.

The DBC.TDWMExceptionLog table contains a row for each time Teradata DWM detects an exception on a query; there could be multiple rows for one query. If Teradata DWM rejects a logon or query, it will be logged. (The SQL of the query with an exception, however, is logged in DBC.SQLTbl.)

You can allow different exception actions for different exception criteria. For example, you can set up Teradata DWM to continue, change WDs, raise alerts, run programs, abort, or even abort only queries using only SELECT statements when it encounters exceptions. Exception directives can even change when OpEnvs change.

Use the Teradata DWM administrator to set the Exception Interval and control how often to activate the asynchronous check for query exceptions. This interval also controls when Teradata DWM checks for (TDWM) delayed blockers.

Note: If you change the MonSesCPUNormalization flag of the DBSControl record, adjust CPU-related exception thresholds. Otherwise, exceptions may not be detected as expected. For more information, see MonSesCPUNormalization of "DBS Control Utility" in *Utilities*.

For more information on exceptions, see *Teradata Dynamic Workload Manager User Guide*.

Managing the System Through a State Matrix

Carefully plan how you want your system to behave when certain events such as time of day, critical applications executing, or resource availability occurs through a state matrix. The state matrix is a two-dimensional diagram that can help you visualize how you want the system to behave in different situations. The matrix is made up of two dimensions:

- System Condition (SysCon) - the condition or health of the system. For example, SysCons include system performance and availability considerations such as number of AMPs in flow control or number of nodes down at system startup.
- Operating Environment (OpEnv) - the kind of work the system is expected to perform. It is usually indicative of time periods or workload windows when particular critical applications such as a crucial data load or month end jobs are running.

The combination of a SysCon and an OpEnv reference a specific "state" of the system. Associated with each state are a set of workload management behaviors, such as throttle thresholds, Priority Scheduler weights, and so on. When specific events occur, they can direct a change of SysCon or OpEnv, resulting in a state change and therefore an adjustment of workload management behavior.

	Operating Environment (OpEnv)			
System Condition (SysCon)		**ALWAYS** (Default)	**LOAD WINDOW** (9 p.m. to 4a.m.)[a]	**END OF MONTH REPORTING**
	NORMAL	Base	LoadState	Throttleback
	BUSY	Throttleback	LoadState	RestrictState
	DEGRADED	RestrictState	Throttleback	RestrictState

[a] Teradata Dynamic Workload Manager allows you to set up event-based management rather than only time-based management. For example, you can define your system to finish jobs for the Load Window regardless of if the time is 4:01 a.m. or later. That is, you can manage workloads based on events rather only by static time windows.

Teradata DWM automatically provides Normal, Always and Base as defaults. In the example state matrix table above, adding a LOAD WINDOW OpEnv means that the ALWAYS OpEnv is active for time outside of the 9 p.m. to 4 a.m. window. Adding the END OF MONTH REPORTING OpEnv, also narrows the time that the ALWAYS OpEnv will be active.

OpEnvs are defined with a precedence; SysCons are defined with a severity. The matrix shows the OpEnvs and SysCons in precedence and severity order. The highest precedence for the “active” OpEnv combined with the highest severity for the “active” SysCon determines the state that Teradata DWM enforces.

For example, assume on a Tuesday at 9 a.m. both the ALWAYS and END OF MONTH REPORTING OpEnvs are active. Since END OF MONTH REPORTING has a higher precedence than ALWAYS, END OF MONTH REPORTING will be used. At the same time, the BUSY and DEGRADED SysCons are both active. Since DEGRADED has a higher severity than BUSY, DEGRADED will be used. The active state will be the state defined from the combination of END OF MONTH REPORTING & DEGRADED which is RestrictState.

To continue the example, say the following throttle rule is defined with different settings per state:

State	Query Limit
Base	100
LoadState	85
Throttleback	50
RestrictState	15

Throttle rules can have different query limits in different states. Because the state is currently at RestrictState, the current limit on the number of queries allowed is 15. Teradata DWM will enforce the limit specified for the current state.

Events

Rules can be associated with the occurrence of event combinations that can trigger a SysCon or OpEnv to become active, or just trigger a notification. Teradata DWM monitors for the following individual events:

- AMP Fatal: Number of AMPs reported as fatal at system startup
- AWT Limit: The number of in-use AWTs on a *x* number of AMPs over *y* duration
- Flow Control: Number of AMPs in flow control
- Gateway Fatal: The number of Gateways reported as fatal at system startup
- Node Down as % of clique: The percent of nodes down in a clique
- PE Fatal: The number of PEs reported as fatal at system startup

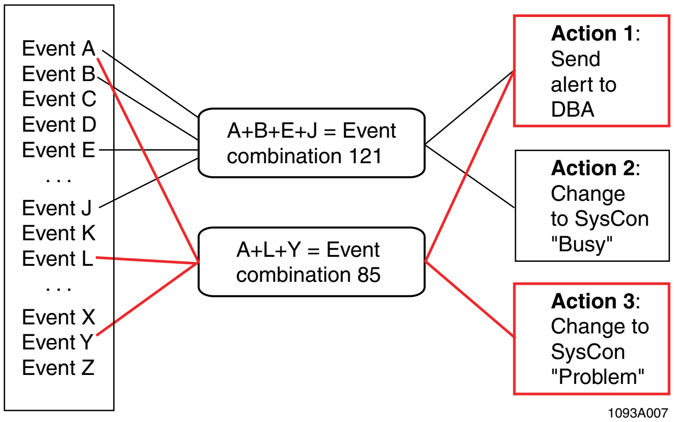
An Event is made up of the threshold value (Limit). For AWT limit and flow control, you additionally specify how long (Qualification Time) the event must be at the threshold level to trigger the event. For example, say an event criteria has been defined as when the AWT Limit exceeds 40 for 5 AMPs for 5 minutes. This event combination is defined with just that event with an action to send an event and an alert and activate the appropriate SysCons when it occurs.

Any logical combination of events is supported.

The actions that an event combination can perform when triggered include:

- Send an alert using Teradata Manager
- Run a specific program with Teradata Manager
- Write to a queue table (SystemQTbl)
- Activate an OpEnv or a SysCon

One single event could be used in different event combinations that result in different actions.


Event A
Event B
Event C
Event D
Event E
. . .
Event J
Event K
Event L
. . .
Event X
Event Y
Event Z
A+B+E+J = Event combination 121
A+L+Y = Event combination 85
Action 1: Send alert to DBA
Action 2: Change to SysCon "Busy"
Action 3: Change to SysCon "Problem"
1093A007

For example, EventA might be acceptable on its own, but still result in a simple "WARNING" alert. But if EventA occurs when another crucial event occurs, the situation can trigger a SysCon to result in a different and automated course of action.

User-Defined Events

You can define your own custom event. When doing so, you must set up these events to be activated and deactivated through the EVENT CONTROL PM/API so that Teradata DWM can detect them. This PM/API allows your applications to participate in workload management decisions. For example, a Dual Active application.

For example, say you have a Dual Active application and it detects System2 is down so that it must move critical applications from System2 to System1. For System1 to be able to handle the load it must be in the “Blue” SysCon. The Dual Active application issues a request to activate the Dual_Down user-defined event (and consequently a corresponding Dual_Down event combination). The Dual_Down event combination has the action to activate the “Blue” SysCon. If “Blue” is the highest severity SysCon, Teradata DWM will switch the current state to the state for the combination of Blue SysCon and the current OpEnv so that the increased load can be better managed.

For more information on the EVENT CONTROL PM/API, see *Workload Management API: PM/API and Open API.*

Example

The following example shows how all the Teradata DWM components work together. Assume the following is defined:

- A user-defined event called “LoadStart” is activated by an API call from an application when data loading starts. It is disabled by another API call when loading completes.
- An Event Combination called “Loading” contains the user-defined “LoadStart” event and when becoming active, has the action to change OpEnv to “Load Window”.
- A state defined as “BatchState” is active when data is loaded in nightly and becomes inactive when all the data is done loading. This load completes not according to a set time but when all the source data becomes available.
- An application named WDX must represent the data as of a certain date.
- A state matrix has a “Load Window” OpEnv and a “Normal” SysCon that map to “BatchState”. In “BatchState”, the system throttles back the WDX application to 0 queries (that is, no queries are allowed to run during the load).

When "LoadStart" is activated, "Loading" becomes active, causing a system state change to "BatchState". During "BatchState", queries from the WDX application cannot run.

After the load is complete and the "LoadStart" event is made inactive, the OpEnv and state changes and the WDX application can resume submitting queries. If WDX had been allowed to run during the load update, there is a chance it could have read some rows that had been updated for the load of the new day while other rows still represented the load of the previous day.

In the above case, the Load Window does not consistently start at some fixed time such as exactly at 9:00 p.m. It is dependent on when all the source data becomes available and is ready to load (this could be any time). In addition, WDX queries are allowed to run until the loading starts which could be well after 9:00 p.m.

Using Teradata DWM Filters or Classifications with Query Bands

Customize and use a query band to identify the originating source of a query. Application IDs, Client User IDs, Account IDs and Database User IDs exist today as part of the logon string, but it is often not enough to properly distinguish submitted requests into appropriate work group granularities.

Setting different query bands for each type of job allows the requests to be classified into different workloads and therefore run with different workload management directives. Therefore, instead of running a whole entire application under a single workload, Teradata DWM can adjust the workload management behaviors of its requests, enabling better use of system resources.Teradata DWM can also check rule associations against the query bands for each request and look for a match first in the query band set for the transaction and then the query band set for the session. If there is a match, Teradata DWM will activate the rule.

For example, assume there is a filter rule that defines QueryBandName=area and QueryBandValue=west:

IF the transaction query band is defined as...	AND the session query band is defined as...	THEN the rule will...
b=gold; area=west	b=silver; area=north	be activated.
b=gold; area=east	b=silver; area=west	not be activated.
b=gold;	b=silver; area=west	be activated.
b=gold;	b=silver; area=south	not be activated.

Using Teradata DWM, you can associate a query band name-value pair with filter rules. The system will then reject or delay requests based on matching name-value pairs in the query bands associated with the request. For more information on how to do this, see *Teradata Dynamic Workload Manager User Guide*.

Using Open API Functions and Stored Procedures

Customize your application by using functions and stored procedures to do the following:

- Abort or release a request on the TDWM delay queue
- Change the workload to which a session is assigned
- Enable or disable filter or throttle rules
- Enable or disable user-defined events
- Obtain Workload Management information such as:
  - Statistics on objects and workload with throttle limits defined for them
  - Which utilities are active in the system and which are in the delay queue
  - List the workload definitions

You can use SQL to call these functions and stored procedures. For more information, see *Workload Management API: PM/API and Open API*.

Internal Teradata DWM Sessions

DBC/SQL Internal Sessions

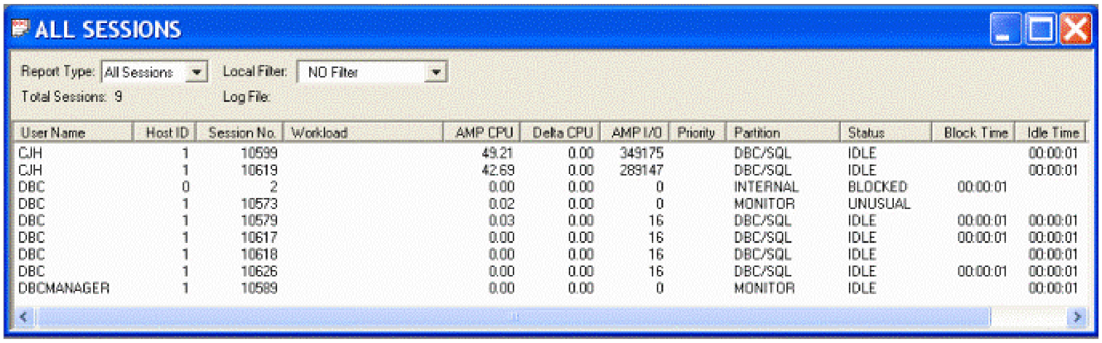
If you enable Teradata DWM logging, Session Information (SI), Performance Monitor (PMON), or the Monitor sessions option in Teradata Manager, they might report an internal session with a hostid of 0 and the partition name "DBC/SQL." This internal session is owned by user TDWM and is used for Teradata DWM processing. It will log off automatically after 20 minutes of inactivity or you can force it off by submitting an ABORT SESSION command with the logoff option using the supervisor window or PM/API. (For more information, see *Workload Management API: PM/API and Open API.*)

Blocked Internal Express Requests

SI, PMON, and the Monitor sessions option in Teradata Manager might also report blocked internal Teradata DWM processing, in the form of express requests, if they cannot complete.

If there is blocked internal Teradata DWM processing, you will see an artificial internal session (HostID = 0 and Session No = 2 with the partition name "INTERNAL"). You would most likely see this particular internal session when you do work, such as table maintenance, that hold locks for a long time and block Teradata DWM internal express requests. If you notice this happening, run your queries when Teradata DWM is not logging or set up maintenance so that locks on the tables are held for a minimal amount of time.

The following Teradata Manager example shows the blocked internal requests identified by the internal session described above:



Report Type: All Sessions
Local Filter: NO Filter
Total Sessions: 9
Log File:

User Name	Host ID	Session No.	Workload	AMP CPU	Delta CPU	AMP I/O	Priority	Partition	Status	Block Time	Idle Time
CJH	1	10599		49.21	0.00	349175		DBC/SQL	IDLE		00:00:01
CJH	1	10619		42.69	0.00	289147		DBC/SQL	IDLE		00:00:01
DBC	0	2		0.00	0.00	0		INTERNAL	BLOCKED	00:00:01	
DBC	1	10573		0.02	0.00	0		MONITOR	UNUSUAL		
DBC	1	10579		0.03	0.00	16		DBC/SQL	IDLE	00:00:01	00:00:01
DBC	1	10617		0.00	0.00	16		DBC/SQL	IDLE	00:00:01	00:00:01
DBC	1	10618		0.00	0.00	16		DBC/SQL	IDLE		00:00:01
DBC	1	10626		0.00	0.00	16		DBC/SQL	IDLE	00:00:01	00:00:01
DBCMANAGER	1	10589		0.00	0.00	0		MONITOR	IDLE		00:00:01

Note: You cannot abort or otherwise administer this internal session.

For more information, see "Handling Blocked Internal Requests That Slow the System" on page 407.

Using Teradata DWM with Queue Tables

When using Teradata DWM with queue tables, be aware of the following:

- Sometimes when a SELECT and CONSUME statement is issued, the row is extracted from the internal queue before satisfying any Teradata DWM rules. No part of the query is executed until it is released by Teradata DWM to run.

- If the SELECT and CONSUME statement is blocked by a rule, the row can be deleted from other sessions if it is being held by the rule for a long time. If the delete does occur, the SELECT and CONSUME will return no row when it is finally processed.
- Also, the other rows in the queue can be consumed from other SELECT and CONSUME statements ahead of the one being held up by the Teradata DWM rules. Thus when that SELECT and CONSUME finally being scheduled is processed, it may have the timestamp much earlier than the other SELECT and CONSUME statement executed earlier. If this occurs, the queue will seem to be processed out of order.

In general, avoid creating Teradata DWM rules that apply to Queue Table.

For more information on queue tables, see “CREATE TABLE (Queue Table Form)” in *SQL Reference: Data Definition Statements*.

For More Information

The following table lists references for more information on using Teradata DWM.

FOR information on how to...	SEE ...
use Teradata DWM to control your query workloads	*Teradata Dynamic Workload Manager User Guide*
define, load, and maintain your query rules and limits	
base workload rules on accounts strings	• “Accounts and Performance Group Names” on page 146. • “Priority Scheduler” in *Utilities*.
start and restart the database	“Chapter 10 Stopping and Restarting the System” on page 283.

Tracking System Object Usage

Database object use counts 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.

You can determine the number of times user queries access or use the following objects[2]:

- Columns
- Databases
- Indexes
- Macros
- Stored Procedures
- Tables
- Triggers
- UDFs
- Users
- Views

[2] Object use count information is not counted for EXPLAIN, INSERT EXPLAIN or DUMP EXPLAIN request modifier.

How Counts Are Collected

When you enable count collection, the system collects use counts from the Optimizer execution plan.

Then it stores counts into cache buffers and writes them to Data Dictionary tables. The ObjectUseCountCollectRate field in the DBS Control record 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 columns AccessCount and LastAccessTimeStamp. (The default value of 0 means that collection is disabled.)

The following table lists the Data Dictionary views that report the count data.

This view...	Provides AccessCount and LastAccessTimeStamp for...
ColumnsX	selected columns.
DatabasesX	selected databases.
IndicesX	hash indexes or join indexes.
TablesX	selected tables, views, macros, stored procedures, triggers, and functions.
Users	users that the requesting user owns or has MODIFY or DROP privileges on.

Note: Storing object counts in the Data Dictionary runs independently of DBQL.

Enabling Count Collection

The object use count data held in cache writes to disk when one of the following happens:

- The value ObjectUseCountCollectRate in DBS Control is reached
- The cache fills up

IF ObjectUseCountCollectRate is...	THEN...
a negative value	a warning message is displayed.
0	the system does not collect object use counts. This is the default.
an integer *n* between 1 and 32767	the system collects object use counts. The Data Dictionary fields AccessCount and LastAccessTimeStamp are updated every *n* minutes. The recommended minimum value is 10 minutes. Any rate below 10 minutes may impact performance of systems with heavy workloads.
a value higher than 32767	a warning message is displayed.

To examine object use counts, use the views listed in the table above in "How Counts Are Collected." For more information on these views, see *Data Dictionary*.

Note: The system reports the counts but you are responsible for deleting and cleaning up unused objects.

Caution: Do not enable count collection if you are performing a dump or restore on the Data Dictionary. Object use count collection is automatically disabled if dictionary locks are held longer than the rate set for ObjectUseCountCollectRate.

Note that a restart causes the information in the cache to be lost and the counts in the Data Dictionary will not be updated with that information.

Usage Recommendations

When collecting object use counts, remember the following:

- Do not enable collection when a dictionary DUMP or RESTORE is in progress.
- The system does not collect counts if it cannot get all dictionary locks (buffer) for the defined collection rate. For example, a dictionary DUMP or RESTORE prevents the system from flushing the cache to the appropriate dictionary fields for a defined collection rate.

  However in the case where an individual dictionary lock for a particular lock is not granted for a defined collection rate, counts are still cached but the dictionary field is not updated and a warning message is written to the DBC.SW_Event_Log table.
- The recommended value of ObjectUseCountCollectRate is 10 minutes or more. Setting ObjectUseCountCollectRate less than 10 minutes impacts system performance.
- Collecting object use counts, like access logging, is resource-intensive. Enable this feature only if necessary, and set the collection rate as needed.

Example 1: 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 (the value of ObjectUseCountCollectRate is a nonzero value).

Now specify some SQL statements 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 TableX to find the use counts for the tables.

```
sel DatabaseName, TableName, AccessCount, LastAccessTimeStamp
from DBC.TablesX where TableName IN ('tab1', 'tab2', 'tab3', 'tab4');
```

The output is as follows:

```
 *** 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 (TVM) have not been accessed since the last resetting of the use counts, the value in the AccessCount column for that object appears as zero, and the LastAccessTimeStamp will appear as a question mark.

Example 2: Object Use Counts for Multi-column Indexes

If a multi-column index is used in a query execution plan, the system records object use counts in the DBC.Indexes table for each column of the index. For example, assume that table t1 has the following definition with a multi-column secondary index on columns c2 and c3:

```
CREATE TABE t1 (c1 INTEGER, c2 INTEGER, c3 INTEGER)
PRIMARY INDEX (c1) INDEX (c2, c3);
```

Also, assume that because of statistics collected on columns c2 and c3, the Optimizer chooses to use the secondary index in the execution plan for the query below:

```
SELECT * FROM t1 WHERE c2 = 2 AND c3 = 3;
```

After the object use count cache flushes and the system writes the data to the Data Dictionary, the DBC.Indices view returns the following information for the secondary index:

```
SELECT IndexNumber, ColumnName, AccessCount, LastAccessTimeStamp
FROM DBC.Indices WHERE TableName = 't1';

IndexNumber ColumnName AccessCount LastAccessTimeStamp
----------- ---------- ----------- -------------------
          4 c2                   1 2006-04-10 09:43:02
          4 c3                   1 2006-04-10 09:43:02
```

Example 3: Object Use Counts for Join Columns

Object use count tallies all column references including join columns. For example, assume that table t1 and t2 have the following definition.

```
CREATE TABLE t1 (a1 INTEGER, a2 INTEGER);
CREATE TABLE t2 (b1 INTEGER, b2 INTEGER);
```

Also, assume that an inner join is performed on the tables using the request below.

```
SELECT a2 FROM t1 INNER JOIN t2 ON a1 = b1;
```

After the object use count cache is flushed, the DBC.Columns view returns the following information for table t1 and t2. Note that the select lists column a2 and the join columns a1 and b1 as counted. Column b2 is not counted because it was not used in the query.

```
SELECT ColumnName, AccessCount, LastAccessTimeStamp
FROM DBC.Columns WHERE TableName IN ('t1', 't2');

ColumnName AccessCount LastAccessTimeStamp
---------- ----------- -------------------
a1                   1 2006-05-15 09:28:20
a2                   1 2006-05-15 09:28:20
b1                   1 2006-05-15 09:28:20
b2                   0                   ?
```

Resetting the Use Count Fields

To clear the use counts, manually update the AccessCount field to zero and the LastAccessTimeStamp field in the Data Dictionary to zero or use the macros described below.

Note: You must have the EXECUTE privilege on the macros in order to use them.

The macro...	resets AccessCount and LastAccessTimeStamp of objects in...
ClearAllDatabaseUseCount	the system.
ClearDatabaseUseCount	a specified database.
ClearTVMUseCount	a specified table, view, macro (or other objects in the TVM table).

To create these macros, use the DIP utility to run the DIPVIEW script if the script has not already been run.

Running the System Initializer (SYSINIT) utility or the upgrade script is required to grant user DBC the “UPDATE” privilege on the AccessCount and LastAccessTimeStamp columns of the Dbase, TVM, TVFields, and Indexes tables.

Note: All SQL statements used by the ClearDatabaseUseCountMacros 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 single quotes, separated by a comma.
the database db2	`Exec DBC.ClearDatabaseUseCount('db2');` Specify databasename in single quotes.
all databases and objects in the system	`Exec DBC.ClearAllDatabaseUseCount;`

Optimizing Performance

For information on optimizing your system resources and how to tune performance, see *Performance Management* for the following topics:

- Collect and use resource usage data to determine bottlenecks
- Set up baseline benchmark profiles for testing
- Set optimal values for DBS Control Record settings
- Use workload and session management to optimize performance
- Tune performance using query analysis resources and tools
- Tune memory to improve overall performance

SECTION 5 Troubleshooting & Caveats

CHAPTER 13 Tracking Processing Behavior with the Database Query Log (DBQL)

The Database Query Log (DBQL) is an optional feature that you can employ to log query processing activity for later analysis. Query counts and response times can be charted and SQL text and processing steps can be compared to fine-tune your applications for optimum performance.

Caution: Logging can be resource intensive and use up database space quickly. If you intend to regularly log queries, you must clean up the DBQL data on a regular basis. This means you will usually examine and send only the analyzed information to a user-defined table. Moving the data off DBC and storing it in a different database will help keep DBC from running out of space.

Do *not* off-load data during peak busy times as this may block ongoing queries.

DBQL Overview

DBQL collects information based on rules you specify and flushes the DBQL cache every 10 minutes[1] and writes to the DBQL dictionary tables (which are a series of predefined tables also referred to as logs). This information includes historical records of queries and their duration, performance data, and target activity.

DBQL is flexible enough to log information on the variety of SQL requests that run on Teradata Database, from short transactions to longer-running analysis and mining queries. You begin and end collection for a user, group of users, account, or a list of accounts.

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, and the times the packets were returned to the host.
- 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

[1] The DBQLFlushRate field in the DBSControl record controls the number of minutes the system will wait before flushing the DBQL cache and writing the collected information into the DBQL tables. For more information, see "Flushing the DBQL Cache" on page 372.

- Detail logging, which includes:
  - Default data
  - Any or all of the following:
    - Step level activity, including parallel steps
    - Object usage per query
    - Full SQL text
    - Explain text

Note: 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. However, other objects that are accessed by the statement will be logged in the DBC.DBQLObjTbl.

Also note that certain optimized requests may bypass the Dispatcher so that DBQL will not log them.

Populating the Log Tables

Like other system tables, the predefined DBQL logs are created as relational tables in database DBC during normal Teradata Database installation. However, while most system tables are populated automatically, you control whether or not to populate the DBQL tables.

If you choose not to use the feature, the tables remain empty. If you want to use the feature, submit a BEGIN QUERY LOGGING statement, with or without options.

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 section “Logging Options” on page 340 describes the options in detail.

The BEGIN/END QUERY LOGGING Statements

DBQL is controlled by the Teradata SQL statements BEGIN QUERY LOGGING and END QUERY LOGGING. Only a user with EXECUTE privilege on DBC.DBQLAccessMacro can invoke the statements.

Note: You cannot issue a BEGIN/END QUERY LOGGING statement while running in ANSI session mode. In Teradata session mode, you cannot issue a BEGIN/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.

For statement syntax, see *SQL Reference: Data Definition Statements*. To enable other users to submit these statements, see “Logging Implementation and Validation” on page 375.

The purpose of each of the Teradata SQL extension to the DCL statements used to enable and disable DBQL logging is described in the following table.

Statement	Purpose
BEGIN QUERY LOGGING	When submitted by a user with EXECUTE privilege on DBQLAccessMacro, this statement enables logging for the named users and accounts. (Log only a maximum of 100 user/account pairs per statement. If logging is to be enabled for more than 100 users, use blocks of BEGIN QUERY LOGGING statements with 100 users each.) For active sessions, logging begins when Teradata Database receives the next query. For more information, see "Logging Scenarios" on page 376.
END QUERY LOGGING	When submitted by a user with EXECUTE privilege on DBQLAccessMacro, this statement stops logging for the named users and accounts. Each END QUERY LOGGING statement should list fewer than 100 user/account string pairs. A routine is called that commits the data and flushes all the DBQL caches. Each time you issue an END QUERY LOGGING statement, the system will flush the cache. Or, you can set up the system to automatically flush the DBQL cache at certain intervals using the DBQLFlushRate field of the DBS Control record. The default value is every 10 minutes (600 seconds) but you may change it to any value between 1 and 3600 seconds. For more information, see "Effects of Dynamically Enabling/Disabling Logging on Current Rules" on page 374.

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').

Logging Users With Multiple Accounts

Only one of the following types of logging levels should be enabled at a time:

- Specific user, specific account
- Specific user, all accounts
- All users, specific accounts
- All users, all accounts

As discussed in "Defining Accounts" on page 143, sessions are always associated with an account. At logon, the session is associated with the default account of the user unless the logon or startup string specifies a different account to which the user is assigned.

Assume user1 has a default account of "ABC" but is also assigned to account "DEF." Assume also that you have submitted the two following logging statements:

RULE 1: *`BEGIN QUERY LOGGING ON ALL ACCOUNT = ('ABC');`*

RULE 2: *`BEGIN QUERY LOGGING WITH OBJECTS ON USER1 ACCOUNT =('DEF');`*

Note: The parentheses are optional for "ACCOUNT =" when there is only one account.

IF user1 logs on...	THEN Teradata Database system uses...
without defining which account he is logging under, for example: `.LOGON user1`	RULE 1.
specifying account DEF, for example: `.LOGON user1, ‘DEF’`	RULE 2.

If you do not specify an account string upon log on, the ON ALL rule could be invoked instead of the ON USER1 rule because the implied default account *happens* to match the ALL rule. To avoid this, either create rules at only one level of logging or always specify an account string in your DBQL rules.

Logging Options

The options to the BEGIN QUERY LOGGING statement are listed in the following table.

Parameter	Logging Behavior
LIMIT SQLTEXT	This option determines how much SQL text is captured for queries logged to the DBQLogTbl. 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 DBQLogTbl completely, specify 0 (zero). Specify LIMIT SQLTEXT=*n* where *n* is some value greater than 0 to control the amount of SQL text to be recorded in DBC.DBQLogTbl. For example, to log 500 characters of the SQL, submit the following: `BEGIN QUERY LOGGING LIMIT SQLTEXT=500 ON USER2;` The maximum limit is 10,000 characters. If you specify the option keyword but not a value, up to 10,000 characters are logged in DBQLogTbl. 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.

<table>
<tr><th>Parameter</th><th>Logging Behavior</th></tr>
<tr><td>LIMIT SUMMARY</td><td>Use this option to count queries based on specified time, AMP CPU, or I/O differentials and store the count results in DBQLSummaryTbl.

SUMMARY is useful for tracking voluminous short queries, such as for OLTP applications, because it does not log into the DBQLogTbl. For an output example, see “Summary Log Table: DBQLSummaryTbl” on page 365.

The LIMIT SUMMARY option:
• Does not generate default rows in DBQLogTbl.
• 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.
You can define the intervals for LIMIT SUMMARY with one of the following:
<table>
<tr><th>Use the unit of...</th><th>To specify summary data as based on...</th></tr>
<tr><td>CPUTIME</td><td>AMP CPU time where n1,n2,n3 is specified in hundredths of a second.

<b>Note:</b> Parser CPU Time is reported, but it is not used to place query data into summary buckets.</td></tr>
<tr><td>CPUTIMENORM</td><td>normalized AMP CPU time where n1,n2,n3 is specified in hundredths of a second.</td></tr>
<tr><td>ELAPSEDSEC</td><td>elapsed time in seconds. This is the default if no qualifier is specified.</td></tr>
<tr><td>ELAPSEDTIME</td><td>elapsed time where n1,n2,n3 is specified in hundredths of a second.</td></tr>
<tr><td>IOCOUNT</td><td>count of total I/O where n1, n2, n3 is specified in number of I/Os.</td></tr>
</table>
If you do not specify units, threshold is based on elapsed time where n1,n2,n3 is specified as elapsed seconds.

For example, to group queries into buckets 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 buckets: 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. The option ELAPSEDTIME provides better granularity for specifying elapsed time in hundredths of a second. This is useful for specifying partial seconds.

Or, to group queries based on I/O, submit:

BEGIN QUERY LOGGING LIMIT SUMMARY=1000,5000,10000 IOCOUNT ON ALL;
For more information on the DBQL Summary table, see “Summary Log Table: DBQLSummaryTbl” on page 365.</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Logging Behavior</th></tr>
<tr><td>LIMIT THRESHOLD</td><td>Queries that run in *n* “units” or less are counted in the DBQLSummaryTbl. Queries that run over n units are logged in the DBQLogTbl. Units can be in one of the following:
<table>
<tr><th>Use the unit of...</th><th>To specify threshold data as based on...</th></tr>
<tr><td>CPUTIME</td><td>AMP CPU time where n is specified in hundredths of a second.
Note: Parser CPU Time is reported, but it is not used to place query data into summary buckets.</td></tr>
<tr><td>CPUTIMENORM</td><td>normalized AMP CPU time where n is specified in hundredths of a second.</td></tr>
<tr><td>ELAPSEDSEC</td><td>elapsed time in seconds. This is the default if no qualifier is specified.</td></tr>
<tr><td>ELAPSEDTIME</td><td>elapsed time where n is specified in hundredths of a second.</td></tr>
<tr><td>IOCOUNT</td><td>count of total I/O where n is specified in number of I/Os.</td></tr>
</table>
If you do not specify units, threshold is based on elapsed time where n is specified as elapsed seconds. The option ELAPSEDTIME provides better granularity for specifying elapsed time in hundredths of a second. This 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.
Note: To count queries using normalized CPU time rather than “raw” CPU time, use the CPUTIMENORM modifier.
To base the threshold on the number of I/Os and log all queries with more than 5000 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).</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Logging Behavior</th></tr>
<tr><td>LIMIT THRESHOLD (Continued)</td><td>
<p>Defining a threshold will determine whether to log a query or just count it, as follows:</p>
<table>
<tr><th>IF a query ...</th><th>THEN DBQL ...</th></tr>
<tr><td>completes at or under the threshold elapsed time, CPU seconds, or I/O count</td><td>
<ul>
<li>Increments the query count and adds the query's elapsed time, CPU time and I/O counts to the summary row for this session in the current collection period.</li>
<li>Stores the final count for the session as a row in DBQLSummaryTbl.</li>
<li>In the summary row, sets the value in the LowHist field to the THRESHOLD time and in the HighHist field to 0 (to identify it as a THRESHOLD row).</li>
</ul></td></tr>
<tr><td>runs beyond the threshold value</td><td>logs a default row for the query in DBQLogTbl so you can examine its execution time through time stamps and the number and level of processing steps.</td></tr>
</table>
<p>Each summary row summarizes the query count, elapsed time, I/O count and CPU time during the collection period regardless if SUMMARY or THRESHOLD criteria are specified.</p>
<p>The criteria determines how the buckets are defined. However, within each bucket, DBQL reports the amount of CPU and I/O that took place during the 10 minute cache flush interval.</p>
<p>For more information, see “Summary Log Table: DBQLSummaryTbl” on page 365.</p>
</td></tr>
<tr><td>WITH ALL</td><td>
<p>WITH ALL logs the information generated by all the WITH rules (EXPLAIN, OBJECT, SQL, and STEPINFO). No other WITH rule is necessary, because ALL generates:</p>
<ul>
<li>One default row per query in DBQLogTbl that includes the first 200 characters of the SQL statement, unless you define LIMIT SQLTEXT=0.</li>
<li>One row per target object per query in DBQLObjTbl.</li>
<li>One row per step per query in DBQLStepTbl.</li>
<li>One or more rows per SQL statement in the DBQLSQLTbl.</li>
<li>One or more rows per query in DBQLExplainTbl.</li>
</ul>
<p><b>Caution:</b> Use this 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>This option cannot be used with SUMMARY or THRESHOLD.</p>
</td></tr>
<tr><td>WITH EXPLAIN</td><td>The DBQLExplainTbl is populated with unformatted explain text. If the additional Explain text is greater than 64KB, multiple rows are generated and the ExpRowNo field indicates the row number.</td></tr>
</table>

Parameter	Logging Behavior
WITH OBJECTS	This option inserts: • A default row in DBQLogTbl • One row per target object per query in DBQLObjTbl Use this option selectively. Object data is useful for analyzing queries that make heavy use of join indexes and indexed access, but can generate many rows. This option *cannot* be used with SUMMARY or THRESHOLD. **Note:** 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. This means, for example, that statements like CREATE TABLE or SELECT FROM DBC.xxx do not have objects logged through DBQL because all objects generated for the query are DBC tables and columns. 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.
WITH SQL	This option inserts: • A default row in DBQLogTbl. • The entire SQL statement for each request for each user being logged. Large statements can cause multiple rows to be written in order to log the full query text. This option cannot be used with SUMMARY or THRESHOLD. 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. **Note:** Set LIMIT SQLTEXT=0 if you specify the WITH SQL option to avoid duplicate logs in DBQLogTbl.
WITH STEPINFO	Populates DBQLStepTbl with AMP step-level information for all SQL queries performed by the specified user. When the query completes, it logs one row for each query step, including parallel steps.

For more information on the SQL syntax, see “BEGIN QUERY LOGGING” in *SQL Reference: Data Definition Statements*.

The DBQL Components

The DBQL logs are a series of system tables created in database DBC during the Teradata Database installation process. The suite of DBQL components includes a security macro and a view for each table, which are created in database DBC by the DIP utility during installation. For details, see "Database Initialization Program (DIP)" in *Utilities*.

The purpose of each object is listed in the following table.

THIS dictionary object ...	IS a ...	THAT ...	AND is created by...
DBQLogTbl	table	stores default rows.	Tableinit (invoked by the Sysinit utility), during installation. **Note:** Because Sysinit clears all data and reformats the disks, do not run it after data rows are loaded.
DBQLObjTbl	table	stores information on the target objects of the query being logged. One row is logged for each object referenced in the query.	Tableinit and only populated if you specify the OBJECT option.
DBQLRuleCountTbl	table	stores the cardinality of DBQLRuleTbl (for internal use only).	Tableinit (invoked by the Sysinit utility). These tables are only populated if you specify BEGIN and END logging statements.
DBQLRuleTbl	table	stores the rules resulting from each BEGIN QUERY LOGGING statement. The END QUERY LOGGING statement removes rows from the rule table. One row exists for each user/ account pair specified in the BEGIN QUERY LOGGING statement and not ended on the END QUERY LOGGING.	Tableinit (invoked by the Sysinit utility). These tables are only populated if you specify BEGIN and END logging statements.
DBQLSQLTbl	table	stores the full SQL text of the query. One query string may require more than one row.	Tableinit and only populated if you specify the SQL option.
DBQLStepTbl	table	stores information about each processing step used to satisfy the query. One row is logged for each step.	Tableinit and only populated if you specify the STEPINFO option.
DBQLSummaryTbl	table	stores queries that meet the criteria for a rule specifying the SUMMARY or THRESHOLD option.	Tableinit (invoked by the Sysinit utility), during installation.
DBQLExplainTbl	table	contains the explain information in an unformatted string without line breaks.	Tableinit and only populated if you specify the EXPLAIN option.

THIS dictionary object ...	IS a ...	THAT ...	AND is created by...
DBQLAccessMacro	macro (empty)	controls authority of users to execute the Teradata SQL BEGIN/END QUERY LOGGING statements.	the DIP utility when it runs the DIPVIEW script.
DBQLRules	view[a]	displays the current rules in DBC.DBCQLRuleTbl (to a user with DBC or SystemFE privileges).	the DIP utility when it runs the DIPVIEW script.
QryLog	view	accesses the DBQLogTbl.	
QryLogExplain	view	accesses the DBQLExplainTbl.	
QryLogObjects	view	accesses the DBQLObjTbl.	
QryLogSummary	view	accesses the DBQLSummaryTbl.	
QryLogSteps	view	accesses the DBQLStepTbl.	
QryLogSQL	view	accesses the DBQLSQLTbl.	
QryLogTDWM	view	accesses the DBQLogTbl.	

[a] The Unicode versions of the DBQL views appear with a V following the view name. For example, QryLogObjectsV is the Unicode version of the QryLogObjects view.

The views listed above have been created for your use. However, you can always create your own customized views by taking some or all of the data from one or more of the DBQL tables and display the information in whatever format you need.

Using QueryID for Joins on DBQL Tables

In previous releases, if the system restarted frequently or data was kept a very long time, the values for QueryID could potentially repeat.The DBQL Option in the DBSControl allowed DBQL to log queries using a longer and more unique QueryID to avoid repeated values.

The less unique form would appear as follows:

```
sel procid, queryid from dbqlogtbl;

ProcID      QueryID
---------   --------------
16383       0016
16383       0066
```

With Teradata Database 12.0 and onward, QueryID is automatically longer and appears system-wide uniquely as follows:

```
sel procid, queryid from dbqlogtbl;
ProcID     QueryID
--------- ------------------
16383      163833600772666007
16383      163833600772666008
16382      163823600772665001
16382      163823600772665002
16382      163823600772665003
16382      163823600772665004
```

System-wide uniqueness for QueryID allows you to do joins using only the QueryID. ProcID is not needed as a join field.

Main Log Table: DBQLogTbl

Default rows are stored in the DBC.DBQLogTbl table, the foundation of the DBQL feature. The default is one default row per query.

Use the LIMIT SUMMARY and LIMIT THRESHOLD options to control whether a default row is logged for a query. Using:

- LIMIT THRESHOLD reduces the amount of logged rows to one row for each query above the threshold limit. Other queries are counted in the DBQLSummaryTbl.
- LIMIT SUMMARY eliminates default row logging and counts all queries in four "buckets" based on the limit criteria.

In addition to the default row logging, you may specify a WITH option to log OBJECTS, SQLTEXT, STEPINFO and/or EXPLAIN. (However, the WITH options cannot be used with SUMMARY or THRESHOLD.)

Note: If you specify the WITH options that result in more detailed information, a default row is still generated in DBQLogTbl.

The Default Row

The fields of the default row provide general query information that is usually adequate for investigating a query that is interfering with performance. When no options are specified, a default row includes:

- User ID and user name under which the session being logged was initiated
- Unique ID for the process, session, and host (client) connection
- Account string, expanded as appropriate, that was current when the query completed
- First 200 characters of the query SQL statement
- CPU and I/O statistics
- Default database name that was current when the query completed

Row Values

The available fields in a DBQLogTbl row are listed in the following table.

DBC.DBQLogTbl Field	Description
ProcID	Unique processor ID of the dispatcher and part of the multi-column NUPI for the table (see also CollectTimeStamp).
CollectTimeStamp	A date and time unique to each buffer cache, which changes for each new buffer. Part of the multi-column NUPI for the table (see also ProcID). This time will be set when the cache is ready to receive new data, not when rows are written to the database.

DBC.DBQLogTbl Field	Description
QueryID	Internally generated identifier of the query and the FK to other DBQL tables. **Note:** QueryID is a system-wide unique field; You can use QueryID to join DBQL tables or TDWM tables with DBQL tables without needing ProcID as an additional join field.
UserID	The unique ID of the user whose query is logged.
AcctString	The account string at the time the row was committed. Because accounts can be changed dynamically at the query or session level (see "SET SESSION ACCOUNT" under "Viewing Account Strings and Session Statistics" on page 147), this may not be the same account that: • DBQL verified for the user at logon time • Was in effect when processing of the query began
ExpandAcctString	The expanded account name under which the query is submitted if account expansion is invoked. **Note:** The &I, which gives the host, session, and request number for the query is different for a stored procedure call. For stored procedure calls, the request number is the client request number which is the request of the CALL statement.
SessionID	Unique session identifier.
LogicalHostID	Unique identifier of the logon source (client connection). A value of 0 indicates the console (PDN).
RequestNum	Unique identifier of the query. **Note:** The request number for the commands that occur within a stored procedure CALL statement will all appear as the same number of the CALL statement itself. For more information on request numbers for stored procedures, see note for ExpandAcctString.
InternalRequestNum	The internal request number. For commands, other than those within a stored procedure, the internal request number and the number in the RequestNum field will be the same. For stored procedures invoked within a session, the internal request number increments by 1 for every request made by the stored procedure. This means that RequestNum will be the request number of the CALL and the value in InternalRequestNum will continue to increment for all other queries issued by the session.
LogonDateTime	The timestamp of the start of the session.
AcctStringTime	Value of the expanded &T ASE code in the account string. If none exists, the value is denoted with a "?" and is null. Even if ASE truncates the expanded account string, DBQL will report the full value.

DBC.DBQLogTbl Field	Description
AcctStringHour	Value of the expanded &H ASE code in the account string. If none exists, the value is denoted with a "?" and is null. Even if ASE truncates the expanded account string, DBQL will report the full value. For example, if &H is in the account definition but does not appear in the expanded account string because it is beyond position 30, DBQL will still report the AcctStringHour value.
AcctStringDate	Value of the expanded &D ASE code in the account string. If none exists, the value is denoted with a "?" and is null. Even if ASE truncates the expanded account string, DBQL will report the full value. For example, if &D is expanded into position 26 or higher in the account definition and truncation occurs, DBQL will still report the entire value of the date in the AcctStringDate field.
LogonSource	The logon source string text.
AppID	Application (executable) or utility name; for example, BTEQ from LogonSource for network clients.
ClientID	The network client user name under which this session is logged (for example, ADMIN) and the PK/FK for the table.
ClientAddr	The client IP address of the submitted query as obtained from Logonsource from a network client.
QueryBand	The query band under which the query is submitted.
ProfileID	The name of the profile, if any, under which the user submitted the query.
StartTime	The timestamp when the query is submitted.
FirstStepTime	The timestamp when the first step is dispatched.
FirstRespTime	The timestamp when the first response packet is sent to host. **Note:** LastRespTime column was removed in Teradata Database 12.0. The DBS only knows when the first response is sent.
LastStateChange	Time of the last State change via Teradata Dynamic Workload Manager.
NumSteps	Total number of Level 1 steps required to process the query. If steps do not apply, such as if the query was aborted or you ended logging for the user before steps were generated, the value is 0 (zero).
NumStepswPar	Total number of Level 1 steps with parallel steps.
MaxStepsInPar	The maximum number of Level 2 steps done in parallel for the query.
NumResultRows	The total number of rows returned for the query.
TotalIOCount	Total I/O used by the query.
AMPCPUTime	Total AMP CPU time used for the query. **Note:** Prior to Teradata Database 12.0, this column was named TotalCPUTime.

DBC.DBQLogTbl Field	Description
ParserCPUTime	Total parser and dispatcher CPU time used for the query.
UtilityByteCount	The number of bytes transferred by MultiLoad or FastLoad insertion, updates, or deletions.
UtilityRowCount	The number of rows inserted, updated, and deleted by Multiload or FastLoad for target tables.
ErrorCode	If greater than 0, this field is the number of the error message in ErrorText. This field contains a code if the query caused a parser syntax error.
ErrorText	If not null, this field provides processing errors and parser errors. For example, the text would explain that the query was not processed due to a parser syntax error.
WarningOnly	Indicator if the error was only a warning. That is, warning mode.
DelayTime	The time a query was delayed by Teradata DWM.
Abort Flag	T if the query being logged was aborted. The collected data could be incomplete.
CacheFlag	This field is blank if the query is not found in step cache. It could also be one of the following: • "T" if the query is found in step cache. • "S" if the query is a parameterized query and a SpecificPlan is generated. • "G" if the query is parameterized query and a GenericPlan is generated. • "A" if the query is parameterized query and a SpecAlways decision is taken. That is, each time a query submitted USING values are peeked at and the query is parsed.
StatementType	The type of statement of the query. In a multistatement request, this is the last statement of the request. However, this may not accurately describe the request. For more statement information, see ExtraField5.
QueryText	The first characters of the SQL query. Up to 200 characters are captured by default. If you use the SQLTEXT option, you can specify that the first n characters be captured where n is 0-10000.
NumOfActiveAmps	Number of AMPs active for the query. Use it to compute the average CPU or I/O per AMP.
MaxAMPCPUTime	CPU time of the highest CPU utilized AMP in the query. **Note:** Previous to Teradata Database 12.0, this field was HotAmp1CPU.

DBC.DBQLogTbl Field	Description
MaxCPUAmpNumber	The number of the AMP with the highest CPU activity. **Note:** Previous to Teradata Database 12.0, this field was HotCPUAmpNumber.
MinAmpCPU	CPU time of the lowest CPU utilized AMP in the query.
MaxAmpIO	I/O count of the highest utilized AMP in the query. **Note:** Previous to Teradata Database 12.0, this field was HotAmpIO.
MaxIOAmpNumber	The number of the AMP with the highest I/O activity.
MinAmpIO	I/O count of the lowest I/O utilized AMP in the query.
SpoolUsage	The maximum amount of spool used while processing the Query. If SpoolUsage is zero, then there is no usage. If it is null, the data on usage was invalid.
WDID	The workload definition number assigned to the query.
OpEnvID	The internal identifier of the Operating Environment currently enforced by Teradata Dynamic Workload Manager.
SysConID	The internal identifier of the System Condition currently enforced by Teradata Dynamic Workload Manager.
LSN	The Logon Sequence Number used for a load utility.
NoClassification	Indicates if the query was not classified.
WDOverride	Indicates if the workload definition assignment was overridden.
SLGMet	Indicates if the query met service level goals.
ExceptionValue	Specifies what type of Teradata DWM exception occurred.
FinalWDID	Indicates the workload definition in which the query completed execution.
TDWMEstMaxRows	The estimated maximum row count generated by the Optimizer used to classify a query into a workload definition.
TDWMEstLastRows	The estimated last row count generated by the Optimizer used to classify a query into a workload definition.
TDWMEstTotalTime	The total time estimated by TDWM software (base on Optimizer estimates) and applied to TDWM rules. **Note:** This time is reported in milliseconds and is only reported when Teradata DWM is active.
TDWMAllAmpFlag	Indicates whether one of the steps of the query is an all-amp step. This is used to classify a query into a workload definition.
TDWMConfLevelUsed	Indicates the confidence level used by Teradata DWM to determine what workload definition to classify a query into.

DBC.DBQLogTbl Field	Description
TDWMRuleID	Rule identifier of the query.
UserName	Name under which the session being logged was initiated.
DefaultDatabase	Name of the default database for the query.
AMPCPUTimeNorm	Normalized AMP CPU time for co-existence systems.
ParserCPUTimeNorm	Normalized Parser CPU time for co-existence systems.
MaxAMPCPUTimeNorm	Normalized maximum CPU time for AMP.
MaxCPUAmpNumberNorm	Number of the AMP with the maximum normalized CPU time.
MinAmpCPUTimeNorm	Normalized minimum CPU time for AMP.
EstResultRows	Estimated result rows as generated by the Optimizer.
EstProcTime	Sum of minimum estimated processing time for steps in the query as generated by the Optimizer. **Note:** This time is reported in seconds.
EstMaxRowCount	The largest number of rows handled by a single step in the query, as estimated by the Optimizer.
ExtraField5	If there is a DDL statement in a request, ExtraField5 reports which type: • DDL Alter • DDL Create • DDL Grant If the statement has only one DML statement or multiple DML statements that are all of the same type, ExtraField5 will indicate the type. For example if there are three DELETE statements in a request, ExtraField5 will report: DML Delete Similarly, for requests with individual or multiple Insert, InsertSelect, Update or Select statements, ExtraField5 will report: • DML Insert • DML InsertSelect • DML Update • Select In a multistatement request with different types of DML statements, you will see a list showing the number of statements of each type in the request. For example, a request with one insert and two update statements will appear as: `DML       Del=0    Ins=1    InsSel=0     Upd=2     Sel=0`

Explain Log Table: DBC.DBQLExplainTbl

This table is populated if the WITH EXPLAIN option is requested in addition to the default information row. If the additional Explain text is greater than 64KB, multiple rows are generated.

DBC.DBQLExplainTbl field	Description
ProcID	Unique processor ID of the dispatcher and part of the multi-column NUPI (also see CollectTimeStamp).
CollectTimeStamp	Time that the rows were written to the database. This field is also part of the multi-column NUPI (also see ProcID).
QueryID	Internally-generated identifier of the query and the FK to other DBQL tables. **Note:** QueryID is a system-wide unique field; You can use QueryID to join DBQL tables or TDWM tables with DBQL tables without needing ProcID as an additional join field.
ExpRowNo	Used if Explain text is greater than 64KB and the system generates multiple rows.
ExplainText	Full Explain text segment. The text is not formatted and appears as a single long string. You may need to program a parsing application to look at the text.

Object Log Table: DBQLObjTbl

With the OBJECTS option, DBQL logs one row in DBQLObjTbl for each data object referenced by the query. An object can be a database, data table, column, secondary index, join index, or journal table. (If the object is a secondary index, its number is logged rather than a name.) DBQL gets the use counts from the Optimizer and not the SQL statement itself. The system logs a count of the number of times the object was accessed by the Optimizer.

Note: 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. This means, for example, that statements like CREATE TABLE or SELECT FROM DBC.xxx will not have objects logged through DBQL because they deal with DBC tables and columns.

Row Values

The following table lists the fields populated in an objects row.

DBC.DBQLObjTbl Field	Description
ProcID	Unique processor ID of the dispatcher and part of the multi-column NUPI (see also CollectTimeStamp).
CollectTimeStamp	Time that the rows were written to the database. This field is also part of the multi-column NUPI (see also ProcID).
QueryID	Internally-generated identifier of the query. **Note:** QueryID is a system-wide unique field; You can use QueryID to join DBQL tables or TDWM tables with DBQL tables without needing ProcID as an additional join field.
ObjectDatabaseName	Name of the database that owns the target object.
ObjectTableName	Name of the table or view.
ObjectColumnName	Name of the column. Or if the ObjectType is 'Idx' or Index, this field returns the name of the column associated with the index. Some queries will not provide a name, such as COLLECT STATISTICS. For multi-column indexes, there will be one additional row for each column in the index and each row will have the same object number.
ObjectID	Unique internal identifier of this object.
ObjectNum	Number of the column or secondary index.

<table>
<tr><th>DBC.DBQLObjTbl Field</th><th>Description</th></tr>
<tr><td>ObjectType</td><td>Character code indicating the type of object targeted.
Possible characters and the object each represents are:
<table>
<tr><th>Type Code</th><th>Object</th></tr>
<tr><td>Agg</td><td>User defined aggregate function</td></tr>
<tr><td>AgS</td><td>User defined aggregate STAT function</td></tr>
<tr><td>Aut</td><td>Security Authorization</td></tr>
<tr><td>Col</td><td>Column</td></tr>
<tr><td>DB</td><td>Database</td></tr>
<tr><td>Hix</td><td>Hash index</td></tr>
<tr><td>Idx</td><td>Index. For each index, there is a database name, table name, and a column name. The ObjectId is the identifier of the table and ObjectNum is the number of the index in that table.
For multi-column indexes, there is one row for each column of the index that a query used. For example, if an index consists of three columns and the query uses all three, there will be three rows, each with a different column name.
The column name will be null for an index for statements such as COLLECT STATISTICS, ALTER PROCEDURE, SHOW PROCEDURE, or SELECT COUNT(*).</td></tr>
<tr><td>JIx</td><td>Join index. For each join index, there is a database name and join index name in the ObjectTableName field. For these rows, the ColumnName will indicate a column referred to by the join index.
• ObjectType is ‘JIx’
• ObjectId matches the id of the join index
• ObjectNum is 0.</td></tr>
<tr><td>Jrl</td><td>Journal</td></tr>
<tr><td>Mac</td><td>Macro</td></tr>
<tr><td>NoT</td><td>No Type (Unknown)</td></tr>
<tr><td>SP</td><td>Stored Procedure</td></tr>
<tr><td>Sta</td><td>User defined STAT function</td></tr>
<tr><td>Tab</td><td>Table</td></tr>
<tr><td>TbF</td><td>Table function</td></tr>
<tr><td>Tmp</td><td>Temporary</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>DBC.DBQLObjTbl Field</th><th>Description</th></tr>
<tr><td>ObjectType (continued)</td><td><table>
<tr><th>Type Code</th><th>Object</th></tr>
<tr><td>Trg</td><td>Trigger</td></tr>
<tr><td>UDF</td><td>User Defined Function</td></tr>
<tr><td>UDM</td><td>User Defined Method</td></tr>
<tr><td>UDT</td><td>User Defined Type</td></tr>
<tr><td>Viw</td><td>View</td></tr>
<tr><td>Vol</td><td>Volatile</td></tr>
<tr><td>XSP</td><td>External Stored Procedure</td></tr>
</table></td></tr>
<tr><td>FreqofUse</td><td>Number of times the object was accessed, as determined by the Optimizer, to process the query.</td></tr>
</table>

SQL Log Table: DBQLSQLTbl

The DBQLSQLTbl is populated if you specify the WITH SQL option in addition to default information. The SQL option logs the full statement text, no matter how large, into DBQLSqlTbl. Multiple rows are generated as necessary.

Note: The full SQL text of a CREATE/ALTER/REPLACE PROCEDURE/ FUNCTION is not logged in DBQLSQLTbl when the statement is submitted with the option to not save the source in the database.

Row Values

The following table lists the populated fields in a DBQLSQLTbl row.

DBC.DBQLSQLTbl Field	Description
ProcID	Unique processor ID of the Dispatcher and part of the multi-column NUPI (also see CollectTimeStamp).
CollectTimeStamp	Time that the rows were written to the database. This field is also part of the multi-column NUPI (see also ProcID).
QueryID	Internally-generated identifier of the query. **Note:** QueryID is a system-wide unique field; You can use QueryID to join DBQL tables or TDWM tables with DBQL tables without needing ProcID as an additional join field.
SQLRowNo	Row number of the statement. Value is 1 unless the statement is large and requires multiple rows.
SQLTextInfo	A full SQL text segment, a string of up to approximately 32,000 characters. **Note:** BTEQ has a limited column length. Use Teradata SQL Assistant, formerly known as Queryman, to display longer lengths.

Query Step Information Log Table: DBQLStepTbl

DBQLStepTbl is populated if you specify the STEPINFO option. When the query completes, the system logs one row for each query step, including parallel steps.

Row Values

The following table lists the populated fields in a DBQLStepTbl row.

DBC.DBQStepTbl Field	Description
ProcID	Unique processor ID of the dispatcher and part of the multi-column NUPI (see also CollectTimeStamp).
CollectTimeStamp	Time that the rows were written to the database. This field is also part of the multi-column NUPI (see also ProcID).
QueryID	Internally-generated identifier of the query. **Note:** QueryID is a system-wide unique field; You can use QueryID to join DBQL tables or TDWM tables with DBQL tables without needing ProcID as an additional join field.
StepLev1Num	Step number. If this row is for a parallel step, the number is repeated for as many parallel steps as were used for this step (for examples, see StepLev2Num).
StepLev2Num	If this row is for a parallel step, this is the second-level number of the step that spawned it. For example, if the value of StepLevl1Num is 4 and this row logs the first parallel step for step 4, this value is 1 (and would read 04 01). If this row logs the second row generated for step 4, this value is 2 (and would read 04 02), and so forth. If this row is not a parallel-step row, this value is 0 (zero).
StepName	Abbreviation of the internal name used to identify this step (for example, DEL for a DELETE step).
StepStartTime	Timestamp when the step was sent to the AMP to the nearest micro second.
StepStopTime	Timestamp when the step returned from the AMP to the nearest micro second.
EstProcTime	The estimated processing time as determined by the Optimizer. **Note:** This time is reported in seconds.
EstCPUCost	An estimate of the milliseconds of CPU time for the step as determined by the Optimizer.

DBC.DBQStepTbl Field	Description
CPUTime	CPU used by this step. **Note:** For parallel steps, the CPU values shown in the first parallel step represent usage for the entire set of parallel steps.
IOCount	I/O used by this step. **Note:** For parallel steps, the I/O values shown in the first parallel step represent usage for the entire set of parallel steps.
EstRowCount	The estimated row count as determined by the Optimizer.
RowCount	If StepName is MRM (Merge Row Multiple) or EXE (MultiLoad), RowCount is the number of rows inserted. For all other steps, RowCount is the actual number of rows returned by the step (indicating activity count). **Note:** See RowCount2 for the number of rows updated or loaded by a step. See ExtraField1 for the number of rows deleted.
RowCount2	If StepName is MRM (Merge Row Multiple) or EXE (MultiLoad), RowCount2 is the number of updated rows. If StepName is LFI (FastLoad), RowCount is the number of rows loaded. **Note:** See RowCount for the number of rows inserted or returned by a step. See ExtraField1 for the number of rows deleted.
NumofActiveAMPs	The number of AMPs involved with the step.
MaxAmpCPUTime	CPU time of the highest CPU utilized AMP in the step. **Note:** Previous to Teradata Database 12.0, this field was HotAmp1CPU.
MaxCPUAmpNumber	The number of the AMP with the highest CPU usage for the step. **Note:** Previous to Teradata Database 12.0, this field was HotCPUAmpNumber.
MinAmpCPUTime	CPU time of the lowest CPU utilized AMP in the step. **Note:** Previous to Teradata Database 12.0, this field was LowAmp1CPU.
MaxAmpIO	I/O count of the highest I/O utilized AMP in the step. **Note:** Previous to Teradata Database 12.0, this field was HotAmpIO.
MaxIOAmpNumber	The number of the AMP with the highest I/O usage for the step.
MinAmpIO	I/O count of the lowest I/O utilized AMP in the query.
SpoolUsage	The number of bytes of spool used for the step.
MaxAMPSpool	Highest spool usage on an AMP.
MaxSpoolAmpNumber	Number of the AMP with high spool usage.
MinAMPSpool	Lowest spool usage on an AMP.

DBC.DBQStepTbl Field	Description
StepWD	The Workload Definition in effect for the step.
LSN	The Logon Sequence Number for the utility.
UtilityTableId	The table ID for the utility. This field is always 0.
RowsWComprColumns	Number of rows with compressed columns.
EstIOCost	An estimate of service time in milliseconds for I/O for this step. There is no comparable actual data for this estimate.
EstNetCost	An estimate of the BYNET service time in milliseconds for the step. There is no comparable actual data for this estimate.
EstHRCost	An estimate of other costs for the step. There is no comparable actual data for this estimate.
CPUTimeNorm	Normalized AMP CPU time for co-existence systems.
MaxAmpCPUTimeNorm	Normalized maximum CPU time for AMP.
MaxCPUAmpNumberNorm	Number of the AMP with the maximum normalized CPU time.
MinAmpCPUTimeNorm	Normalized minimum CPU time for AMP.
ExtraField1	If StepName is EXE (MultiLoad), this field is the number of rows deleted. **Note:** This field is called “RowCount3” in the QryLogSteps view. For the number of rows inserted or returned by a step, see RowCount. For the number of rows updated or loaded by a step, see RowCount2.

Step Descriptions

The steps reflected in the DBQLStepTbl are generated by Teradata Database to execute a query. They may not always coincide with steps seen in the Explain because some steps are for internal use only.

The following table lists the possible logged step names and their meanings.

Step Internal Name	Description
AAB	Asynchronous Abort of a DBC/SQL request
ALT	Insert target and originating table IDs into the HUT Control Segment for a Logon Sequence Number
AltGrp	Alter replication group
BMSMS	Bit map set manipulations such as intersect and union
CIX	Create secondary index

Step Internal Name	Description
CkNPK	Check N parent keys
CKP	Checkpoint database
CRI	Create reference index
CrtGrp	Create replication group
CSSUM	Collect Statistics aggregate operations
CSUPD	Collect Statistics update
Ctb	Create table header
CTRts	Create table access rights
DEL	Delete row
DELQT	Delete row from queue table
DIRI	Drop inconsistent reference index
DIX	Delete secondary index
DJT	Delete journal table
DRI	Delete reference index
DRIDR	Delete RID Row
DrpGrp	Drop a replication group
DTB	Drop table
DTmp	Delete temporary tables
EBD	BEGIN EDIT DELETE
EBE	BEGIN MLOAD
EDM	Prepare for deblocker/application task (DML)
Edt	End transaction
EED	Prepare MLOAD work tables
EEN	END edit step
ERE	Release MLOAD work tables
ESD	Data acquisition
ESR	Sort MLOAD work tables
EVT	Create event table row
EXE	Execute edit
ExecSP	Execute Stored Procedure
ExpHR	Export Horizontal Redistribution step

Step Internal Name	Description
ExpRL	Export Release Locks Step
ExpVR	Export Vertical Redistribution step
Fail	Operation did not work
FCF	Forward configuration
FDS	Flush DBSpace accounting table entry
FLGRI	Flag reference index
Hcs	Add table record info and access rights into the Hut Control Segment for a Logon Sequence Number
HLA	High-Level Large Object Access
HUL	Host utility set lock
ILR	Contain access log entries
INS	Insert a row
INSLDC	Insert Deferred Lob Constant
INSQT	Insert a row to a queue table
InvJHI	Invalidate Join/Hash index
JIN	Join
JTmp	Journal temporary tables
LBG	Data-Load BEGIN LOADING
LCM	Load config map
LCP	Data-Load CHECKPOINT LOADING
LFI	Data-Load END LOADING
LIN	Data-Load USING ... INSERT
LobFet	Large object fetch
LogAOf	Logging for online archive off
LogAOn	Logging for online archive on
LogPD	Log Plan Directive
LOT	Large object transport
LOT2VM	Forward LOT to Virtual Mailbox
MDT	Merge delete tables
MiIns	Merge-Into insert

Step Internal Name	Description
MiRet	Merge-Into retrieve There is a Merge-Into table reference in a RET (that is, retrieve) query.
MiUpd	Merge-Into update
MLK	Multi-Lock: several LCK Messages in one
MRD	Merge delete two tables based on row hash code
MRG	Merge two tables based on row hash code
MRL	Merge utility locks
MRU	Merge update
MRM	Merge Row Multiple Operations Step
MTB	Modify table. To modify the table internally, the table header must also be modified
MTH	Modify table header
MTmp	Materialize temporary tables
MVN	Modify version number in table header
NEXTOK	It is okay to send the next step
OAR	One AMP reply
Okay	Operation worked
OKELCT	Send StpOKElicitData message to notify dispatcher that it is fine to send next step within the same request. It is required to prevent deadlocks and serve as a form of flow control.
PFN	Process function
PKA	Prime key abort
PRELOB	InsPreLob step
QryLgn	Query logon on step
RAE	Remove abort entry from TOABORT list
RepPh1	Replication Phase 1 step
RepRCR	Replication replay changed rows message
RET	Retrieve row
RpChg	Subscriber replication change row message
RSF	Release spool files
SAMP	Perform sampling from table/spool
SAT	Synchronous abort test step from DBC/SQL statement

Step Internal Name	Description
SMS	Set manipulations such as intersect, union, minus
SplDB	Spoil database information in Data Dictionary
SplPSC	Spoil parser session information
SplQL	Spoil the DBQL Rule cache
SplRTS	Spoil Stored Procedure cache information
SplTAB	Spoil Table information in Data Dictionary
SQB	Set Query_Band
SQL	SQL step for Request Text storage
SRD	Sort a table
STATFN	Perform statistical functions
StpRLK	Release lock
SUM	Perform local aggregate operations
TRET	Trigger retrieve step
UPD	Update row
UpsIns	Upsert insert
UpsUpd	Upsert update
VJT	Validate journal table
VRQ	2PC vote step
Warn	Warning response

Summary Log Table: DBQLSummaryTbl

DBC.DBQLSummaryTbl holds the counts logged for queries of users with THRESHOLD or SUMMARY rules. It is only populated if summary information is requested or a threshold value is used.

Rows Generated by the THRESHOLD Option

If a threshold value is used, the DBQLSummaryTbl will log queries based on the criterion of elapsed seconds, elapsed hundredths, normalized time, I/O count, or AMP CPU hundredths of a second. The ValueType column in DBQLSummaryTbl tells which criterion are used.

Queries with values less than or equal to the given limit will be counted in the DBQLSummaryTbl. Queries with more than the given limit will be logged in DBQLogTbl.

For example, if the criterion is elapsed seconds, for each session being logged, DBQL with the THRESHOLD option does the following:

- Each time a query completes within the threshold time, increments the counters for the session
- Every 10 minutes, writes the cumulative count for each session as a separate row in DBQLSummaryTbl
- For any query that exceeds the threshold time, DBQL generates a default row in DBQLogTbl

For example, if you specified "LIMIT THRESHOLD=500 CPUTIME ON user1", you could find out how many queries user1 submitted that required more than 5 CPU seconds.

The following is an example of how to query the DBQLSummaryTbl table, using the QryLogSummary view, to find the total I/O count and total CPU time:

```
SELECT collecttimestamp, sessionid, querycount, totalcputime, totaliocount
FROM QryLogSummary ORDER BY collecttimestamp;
```

```
CollectTimeStamp         SessionID       QueryCount       AMPCPUTime    TotalIOCount
------------------   ----------    ------------  ---------  -----------
2006-07-29 10:30:05           1,001           5          .031             4
2006-07-29 11:07:10           1,001           1          .015             0
2006-07-29 11:07:10           1,013           1          .000             0
2006-07-29 11:07:10           1,000           2          .047             0
2006-07-29 11:07:10           1,014          71          .907         2,427
2006-07-29 11:17:10           1,017          96         1.234         2,983
2006-07-29 11:17:10           1,014          26          .329           552
2006-07-29 11:17:10           1,031           1          .016             0
2006-07-29 11:17:10           1,023          94         1.093         2,483
2006-07-29 11:17:10           1,026          42          .578         1,196
```

Rows Generated by the SUMMARY Option

DBQL behavior resulting from the SUMMARY option is unique in that:

- No default rows are generated to DBQLogTbl for summarized queries
- You define the summary criteria and then DBQL summarizes queries into those buckets. Each bucket counts the number of queries for the session that fall into that bucket and also sums up their elapsed time, I/O count and various CPU times.
- DBQL maintains summary counters in cache. The contents are committed to the DBQLSummaryTbl table every 10 minutes, when the cache is flushed

The following table describes the summary criteria that make up the buckets.

Using the following modifier on the SUMMARY Option...	Means the system logs the values specified as...
CPUTIME	hundredths of a second of AMP CPU time. So for example: `BEGIN QUERY LOGGING LIMIT SUMMARY=5, 10, 15 CPUTIME ON user1;` means counting queries based on CPU time where 5 is 0.05 seconds of CPU time, 10 is 0.10 seconds and 15 is 0.15 seconds of CPU time.

Using the following modifier on the SUMMARY Option…	Means the system logs the values specified as…
CPUTIMENORM	hundredths of a second of normalized AMP CPU time. This is similar to CPUTIME above, but uses the times from a co-existence system that are normalized.

ELAPSEDSEC	Without an modifier to the SUMMARY option:
	`BEGIN QUERY LOGGING LIMIT SUMMARY=5, 10, 15 ON user1;`
	elapsed seconds is used. You can explicitly set elapsed seconds as a criteria by using the following statement:
	`BEGIN QUERY LOGGING LIMIT SUMMARY=5, 10, 15 ELAPSEDSEC ON user1;`

ELAPSEDTIME	elapsed time where n is specified in hundredths of a second.
IOCOUNT	the number of I/Os used. For example: `BEGIN QUERY LOGGING LIMIT SUMMARY=1000,5000,10000 IOCOUNT ON ALL;`

DBQL gathers information for the collection interval (the default is every 10 minutes). For every collection interval and every active session at the end of the interval, there is as many as four rows per session containing the query count, the number of seconds of elapsed time for those queries, the amount of CPU time and the number of I/Os. Each row represents one of four possible buckets of information.

In the default case, where elapsed time is used to qualify a query for one of the four buckets, the following buckets are possible:

- 0 to 5 seconds
- Longer than 5 seconds to 10 seconds
- Longer than 10 seconds to 15 seconds
- Longer than 15 seconds

For example, if during that next logging period for the user, two queries ran under 5 seconds, three queries ran for 7 seconds, and one query ran for over 15 seconds, the following rows would be written to DBQLSummaryTbl for the session:

```
COUNT  SECONDS  LOWHIST  HIGHHIST
-----  -------  -------  --------
    2        1        0         5
    3       21        5        10
    1      200       15     32767
```

For this example, there were:

- No queries between 10 and 15 seconds
- To determine the average time for each query counted, divide SECONDS by COUNT (for example, the two queries in the first row averaged 0.5 seconds each; the three queries in the second row averaged 7 seconds each).

If you specified LIMIT THRESHOLD=500 CPUTIME ON user1, you could find out how many queries user1 submitted that required more than 5 CPU seconds.

Row Values

The following table lists the fields available for population in DBQLSummaryTbl rows.

DBQLSummaryTbl Field	Description
ProcID	Unique processor ID of the dispatcher and part of the multi-column NUPI (also see CollectTimeStamp).
CollectTimeStamp	Time that the row was inserted into the database. This field is also part of the multi-column NUPI (also see ProcID).
UserID	The unique ID of the user whose query is logged.
AcctString	The unexpanded account name under which the query is submitted.
LogicalHostID	The logical host from which the query is submitted.
AppID	The application ID under which the query is submitted.
ClientID	The client ID under which the query is submitted.
ClientAddr	The client IP address of the submitted query.
ProfileID	The profile ID under which the user is submitting queries.
SessionID	Identifies the session for which queries are being summarized (counted).
QueryCount	Number of queries run in the last 10 minutes.
ValueType	The THRESHOLD criterion is as follows: • C = AMP CPU seconds • H = elapsed hundredths of a second • I = I/O count • N = Normalized AMP CPU seconds • S = elapsed time in seconds.
QuerySeconds	Total run time of queries in this period for this histogram. Calculated to the nearest hundredth of a second.
TotalIOCount	The total I/O count.
AMPCPUTime	Total AMP CPU time. **Note:** Previous to Teradata Database 12.0. this field was TotalCPUTime.
ParserCPUTime	CPU time to parse queries.
AMPCPUTimeNorm	Normalized AMP CPU time.
ParserCPUTimeNorm	Normalized CPU time to parse queries.

DBQLSummaryTbl Field	Description
HighHist	• Highest value for the SUMMARY option. • 0 (zero) for the THRESHOLD option. • If the modifiers CPUTIME or IOCOUNT are used, then this value is the highest CPU seconds or I/O counts set by the user. Use CPUTIMENORM for hundredths of a second of normalized CPU time. To measure subsecond queries, use ELAPSEDTIME. The default, elapsed time in seconds, can be explicitly specified with ELAPSEDSEC.
LowHist	• Lowest value for the SUMMARY option. • For the THRESHOLD option, this is the threshold time set by the user. The default time is 5. • If the modifiers CPUTIME or IOCOUNT are used, then this value is the lowest CPU seconds or I/O counts set by the user. Use CPUTIMENORM for hundredths of a second of normalized CPU time. To measure subsecond queries, use ELAPSEDTIME. The default, elapsed time in seconds, can be explicitly specified with ELAPSEDSEC.

Things to Consider When Logging DBQL Data

You must have EXECUTE privilege on the DBC.DBQLAccessMacro macro to log DBQL data.

To start logging, specify at least one user or account in a BEGIN QUERY LOGGING statement. Each unique name or name pair creates a rule. You can specify options or not. If you do not, one default row is logged per query. If you do, the tables that are populated depend on your choices. You can specify that the recording criteria be a mix of:

- Users and accounts
- Elapsed time, AMP CPU usage, I/O counts where limits can be expressed as:
  - A series of intervals
  - A threshold limit
- Processing detail, including any or all:
  - Objects
  - Steps
  - SQL text
  - EXPLAIN text

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.

For more information on the BEGIN QUERY LOGGING statement, see *SQL Reference: Data Definition Statements*.

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 this log ...
default (no options specified)	query	one default row per query	DBQLogTbl
individual, because the query ran longer than THRESHOLD seconds	long-running query	one default row per long-running query	
individual, because the query ran more than THRESHOLD usage	query that exceeds CPU or I/O time	one default row per long-running query	
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 response time interval 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

For more information on best practices for data collection and maintenance, see *Performance Management*.

Whom to Log Information For

Once logging has begun for a user or account, rows continue to be generated during every session until you end logging for that user or account. Therefore, be very careful to log information only for the users or accounts you need.

Note: You can only specify 100 name/account combinations in one BEGIN QUERY LOGGING command.

Only one of the following types of logging levels should be enabled at a time:

- One specific user, one specific account. If you do not specify which account, the system uses to the default account defined for that user. You should always explicitly specify which account you want to log for that user. For more information, see "Logging Users With Multiple Accounts" on page 339.
- One specific user, all accounts. (Log only a maximum of 100 user/account pairs per single BEGIN QUERY LOGGING statement. If logging is to be enabled for more than 100 users, use blocks of BEGIN QUERY LOGGING statements with 100 users each.)
- ON ALL users for specific accounts. Any user who logs on using one of the specified accounts will be logged.
- ON ALL users for all accounts. Avoid logging all users, because it logs every query of every session to the DBQLogTbl. You can reduce the amount of rows generated by using the LIMIT THRESHOLD option, but typically all users is only appropriate in special testing environments where growth can be kept to a minimum.

Caution: The results of the ON ALL (users) option can fill up the DBQLogTbl very rapidly. DBQL rows consume PERM space in database DBC and remain until you explicitly delete them (see "Purging the System Logs" on page 120). Be aware that DBC space is required by other system tables and also by your Transient Journal (see "Transient Journal (TJ) and DBC Space" on page 230).

When a BEGIN QUERY LOGGING statement is processed, a row is placed in the DBQLRuleTbl for each user or account string specified. When the logging status of a user is verified, an entry is placed in the rules cache.

Applying the Logged Data

First-level information is captured in DBQLogTbl and DBQLSummaryTbl. For short-running, high-volume queries, you can request the THRESHOLD or SUMMARY 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 DBQLogTbl and DBQLSummaryTbl to identify issues such as workloads that do not meet response time service-level agreements. If you suspect a particular query (for example, a query exceeded the THRESHOLD limit), resubmit it while logging with the WITH SQL option to capture the full SQL text. Then you can replay it while logging WITH OBJECT detail or WITH STEPINFO detail (each time setting SQLTEXT=0).

When used selectively, detail 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, Performance Monitor, ResUsage, and DBC.AMPUsage
- Make efficient use of existing capacity and plan for future expansion by factoring in exponential growth of query demand

You can also input detail data for Baseline Profiling, Target Level Emulation, and Teradata client tools and utilities such as Meta Data Services, Teradata Manager, and Teradata Visual Explain. Client tools aid in analysis and present the information in a graphic form that is easily manipulated and understood. For more information, see Chapter 11: "Recommended Housekeeping Tasks" or *Teradata Visual Explain User Guide*.

Flushing the DBQL Cache

The DBQLFlushRate field in the DBSControl record defines the number of seconds the system waits before flushing the data in each DBQL cache to the DBQL tables. The default is 600 seconds (10 minutes).

For example, if the DBQLFlushRate field is set to 900 seconds (15 minutes), all the cache entries will be written to the DBQL tables at least every 15 minutes. If a cache is filled after 13 minutes, entries will be written at 13 minutes and again at the 15 minute interval.

Note: You can select a rate from 1 to 3600 seconds. However, Teradata recommends a rate no less than 10 minutes. A rate less than 10 minutes can impact performance.

Example of Logging Results

The following table describes examples of logging options and the logged results.

IF you define accounts as follows ...	AND you submit the following statement ...	THEN logging results are ...
`MODIFY PROFILE WebUsers AS ACCOUNT='WebQry&D&H';`	`BEGIN QUERY LOGGING WITH OBJECTS ON ALL ACCOUNT=('WebQry&D&H');`	A row in DBQLObjTbl and DBQLogTbl for each object for each query during each 'WebQry&D&H' session.
`MODIFY PROFILE TactUsers AS ACCOUNT='$HTactQry&D&H';`	`BEGIN QUERY LOGGING WITH STEPINFO ON ALL ACCOUNT=('$HTactQry&D&H');`	A row in DBQLStepTbl and DBQLogTbl for each step of each query during each '$HTactQry&D&H' session.
`MODIFY PROFILE StratUsers AS ACCOUNT='$LStratQry&D&H';`	`BEGIN QUERY LOGGING LIMIT THRESHOLD=3 ON ALL ACCOUNT=('$LStratQry&D&H');`	For each '$LStratQry&D&H' session: • 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 default data in DBQLogTbl.

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         D         1
D_PERSONNEL   DEPARTMENT    ?             00009005   0         T         1
D_PERSONNEL   DEPARTMENT    DeptNo        00009005   1,025     C         2
D_PERSONNEL   DEPARTMENT    DeptName      00009005   1,026     C         1
D_PERSONNEL   DEPARTMENT    EmpCount      00009005   1,027     C         1
D_PERSONNEL   DEPARTMENT    Loc           00009005   1,028     C         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
 2         0         MLK        2004-07-08 20:37:23.770000   2004-07-08 20:37:23.780000    0
 3         0         MLK        2004-07-08 20:37:23.780000   2004-07-08 20:37:23.790000    1
 4         1         SAT        2004-07-08 20:37:23.790000   2004-07-08 20:37:23.790000    0
 4         2         SAT        2004-07-08 20:37:23.790000   2004-07-08 20:37:23.790000    0
 4         3         INS        2004-07-08 20:37:23.800000   2004-07-08 20:37:23.800000    1
 4         4         INS        2004-07-08 20:37:23.800000   2004-07-08 20:37:23.810000    1
 4         5         INS        2004-07-08 20:37:23.820000   2004-07-08 20:37:24.830000    1
 4         6         INS        2004-07-08 20:37:23.830000   2004-07-08 20:37:24.840000    1
 4         7         CTRts      2004-07-08 20:37:24.110000   2004-07-08 20:37:25.060000    1
 5         0         Ctb        2004-07-08 20:37:25.080000   2004-07-08 20:37:25.100000    1
 6         0         Edt        2004-07-08 20:37:25.120000   2004-07-08 20:37:25.130000    1
```

Reviewing or Ending Current Rules

The following sections describe how to review and end DBQL rules.

Reviewing Rules

The DBC.DBQLRules view provides a window into the contents of the DBQLRuleTbl table. Note that only a user with SELECT privilege on DBC.DBQLRules can access the view. A SELECT on the DBQLRules view displays the rules currently in effect. You can qualify the response by user name, or account string (if it exists), or both.

```
SELECT * FROM DBC.DBQLRULESV WHERE ACCOUNTSTRING='$LTest&D&H';
```

The following table describes the fields the DBQLRules view can return.

Field	Description
User name	The name of a user for whom you defined logging rules.
Account	An account for which you defined logging rules, or which you defined as the qualifier for logging sessions of this user.
Logging options (if defined)	Any logging options you defined for this user or account, such as OBJECTS, EXPLAIN, STEPINFO, SQL, SQLTEXT, SUMMARY, or THRESHOLD.

Field	Description
Time parameters (if defined)	Any time intervals or limits you defined for the SUMMARY and THRESHOLD options.
Type of criteria	Specify one of the following criteria with summary and threshold options: • 0 = Elapsed seconds • 1 = AMP CPU (hundredths of a second) • 2 = I/O • 3 = Normalized AMP CPU (hundredths of a second) • 4 = Elapsed hundreds
Size parameter	Number of bytes of text in QueryText in DBQLogTbl.

Ending DBQL Logging for Specific 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 USER1 ACCOUNT='DEF';
```

to stop logging, use:

```
END QUERY LOGGING ON USER1 ACCOUNT='DEF';
```

Teradata recommends using scripts to enable and disable logging to avoid typographical errors.

Effects of Dynamically Enabling/Disabling Logging on Current Rules

When you enable 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	• Data for the current query is not collected. • 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.

IF you...	AND a query for that session is already...	THEN ...
disable logging (submit an END QUERY LOGGING statement) for an active session	cached	• 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.

Logging Implementation and Validation

Implementing Query Logging

You need the EXECUTE privilege on the special macro DBQLAccessMacro to submit the BEGIN QUERY LOGGING statement. DBQLAccessMacro is created by the DIPView script 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 ADMIN, 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 components listed in "The DBQL Components" on page 345 should be reported.

3 Grant the following privileges to your database administrator user:

```
GRANT EXECUTE ON DBC.DBQLAccessMacro TO ADMIN ;
GRANT SELECT ON DBC.DBQLRULES TO ADMIN ;
```

4 Log off the DBC or SystemFE session.

5 Log on again as user ADMIN.

6 Define query logging rules for one or more users or one or more accounts using BEGIN QUERY LOGGING statements. (For full syntax, see *SQL Reference: Data Definition Statements*.)

7 Check the DBQLRules view to see if the rules are correct:

```
SELECT * FROM DBC.DBQLRules ;
```

If you find an error, submit an END QUERY LOGGING statement for that user and define a new BEGIN QUERY LOGGING statement.

Rules Validation

The following table lists the rules validated when DBQL logging has begun.

IF a...	THEN DBQL ...
user logs on to start a session	checks the logon string against the user or account in the rules cache.
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.

Logging Scenarios

At session start, the DBC determines if DBQL is enabled for the session. If logging is enabled, one row is logged for each query (unless you specified the SUMMARY or THRESHOLD option; for an explanation, see "Summary Log Table: DBQLSummaryTbl" on page 365).

The following table offers examples of the type of data that is logged according to the rules you defined and the behavior of the query.

IF you...	THEN ...
submit a BEGIN 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 a BEGIN QUERY LOGGING ON ALL ACCOUNT = 'ABC';	any user that logs on with the account string 'ABC' will be logged.
try to drop a user with DBQL enabled for that user	an error 5780 occurs. You cannot drop a user that has DBQL logging enabled. First disable logging for that user.
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 statement	the BEGIN QUERY LOGGING statement fails with an access error: `Failure 3523 (username does not have statement permission)`

IF you...	THEN ...
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 DBQLogTbl row for the query.
want to view all logged rows for a query	use the QueryID field to join DBQLogTbl rows with (depending on the rules for the user) DBQLObjTbl, DBQLStepTbl, or DBQLSQLTbl rows.
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 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.
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 shows a rule for the user, but queries run by that user are not logged because the sessions never match the user/account pair.
begin query logging for a specific account for a user, and the account includes ASE codes	both the input account string and the expanded account string are logged. (For details on ASE codes, see “Tracking Accounts With Account String Expansion (ASE)” on page 148.) The system automatically strips away leading blanks and converts all letters in an account string to upper case before storing it in DBC.DBQLRuleTbl.
begin query logging for a user and that user runs the same query multiple times during one session	multiple rows are logged in DBQLogTbl. 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.
define a rule for a user specifying OBJECTS	if the user runs a SELECT that joins two tables owned by the same database: • One row for the query is logged in DBQLogTbl • Rows are logged in DBQLObjTbl as follows: • 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 OBJECTS and the user runs a query that causes the Optimizer to reference the same object twice	• One row for the object is logged in DBQLObjTbl. • The value in the FreqofUse field is incremented to 2.

<table>
<tr><th>IF you...</th><th>THEN ...</th></tr>
<tr><td>begin logging for a user with no options and the user runs a query with more than 200 characters</td><td>a row is logged in DBQLogTbl that includes the first 200 characters of the query.</td></tr>
<tr><td>create a rule for a user that specifies SQLTEXT= 0 and the user runs a query</td><td>logging depends on the following:
<table>
<tr><th>IF you ...</th><th>THEN ...</th></tr>
<tr><td>also specified the [WITH] ALL or SQL option</td><td>the SQL text is logged only in DBQLSQLTbl.</td></tr>
<tr><td>did not also specify WITH ALL or SQL</td><td>no SQL characters are logged.</td></tr>
</table></td></tr>
<tr><td>define a rule specifying SQLTEXT=10000 and the user runs a query containing 15000 characters</td><td>a row is logged in DBQLogTbl that includes the first 10,000 SQL characters.</td></tr>
<tr><td>create a rule specifying SQLTEXT=32000 (or anything larger than 10,000) and the user runs a query comprising >31000 SQL characters</td><td>a row is logged in DBQLogTbl that includes the first 10,000 characters.</td></tr>
<tr><td>define a rule with STEPINFO and the user runs a query that does not generate parallel steps</td><td>One row is logged in DBQLStepTbl for each step used to resolve the query. In each row, the value of StepLev2Num is 0.</td></tr>
<tr><td>define a rule with STEPINFO and the user runs a query that generates parallel steps</td><td>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.</td></tr>
<tr><td>define a rule with just the SQL option</td><td>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.</td></tr>
<tr><td>define a rule with both the SQL option and SQLTEXT=1000</td><td>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.</td></tr>
<tr><td>define a rule with the SQL option and SQLTEXT=0</td><td>None of the SQL characters are saved in DBQLogTbl. The entire statement is logged in as many rows as required in DBQLSQLTbl.</td></tr>
<tr><td>create rules for a user and specify SUMMARY=5,10,15 and during the next session, every query takes longer than 5 seconds but less than 10 seconds to complete</td><td>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.</td></tr>
<tr><td>create rules for a user and specify SUMMARY = 5,10,15 and during the next session every query completes in less than 5 seconds</td><td>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.</td></tr>
</table>

IF you...	THEN ...
create rules and specify 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 THRESHOLD only (without a time value), and UserA then processes four queries, where:
• 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:
	• The longer query is logged fully in DBQLogTbl, with values in all valid fields of the row
	• For each of the three shorter queries:
	  • No entries are made in DBQLogTbl
	  • DBQLSummaryTbl will have a row with these values:
	`COUNT = 3`
	`SECONDS = 10`
	`LOWHIST = 5`
	`HIGHHIST = 0`

create a rule and specify THRESHOLD=100000	An error is returned; THRESHOLD must be less than 32K.

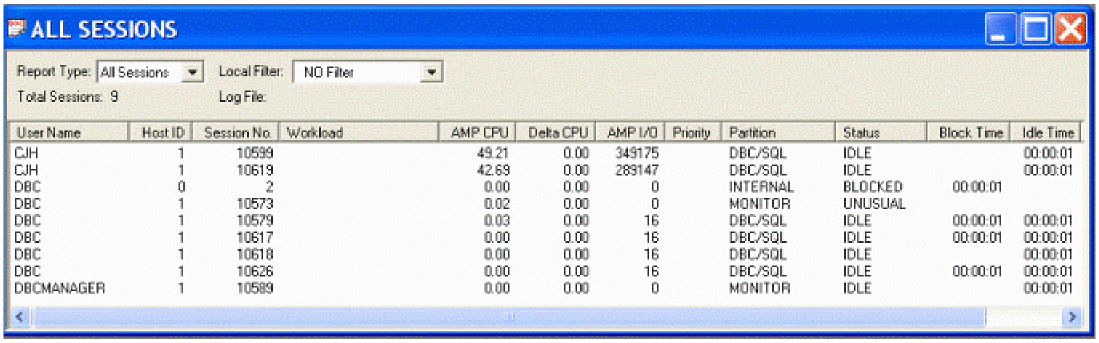
Handling Blocked Internal DBQL Requests

Session Information (SI), Performance Monitor (PMON), and the Monitor sessions option in Teradata Manager all report when internal DBQL processing, in the form of express requests, cannot complete and are blocked.

If there is blocked internal DBQL processing, you will see an artificial internal session number (HostID = 0 and Session No = 2 with the partition name “INTERNAL”). You would most likely see this particular internal session when you do work, such as table maintenance, that hold locks for a long time and block DBQL internal express requests. If you notice this happening, run your queries when there is no DBQL logging occurring. When you must access the tables, make sure the locks on the tables are held for a minimal amount of time.

Note: You *must* submit an END QUERY LOGGING statement for any previously established query logging before you do maintenance on DBQL tables.

The following Teradata Manager example shows the blocked internal requests identified by the word “INTERNAL” under the Partition column:



Report Type: All Sessions
Local Filter: NO Filter
Total Sessions: 9
Log File:

User Name	Host ID	Session No.	Workload	AMP CPU	Delta CPU	AMP I/O	Priority	Partition	Status	Block Time	Idle Time
CJH	1	10599		49.21	0.00	349175		DBC/SQL	IDLE		00:00:01
CJH	1	10619		42.69	0.00	289147		DBC/SQL	IDLE		00:00:01
DBC	0	2		0.00	0.00	0		INTERNAL	BLOCKED	00:00:01	
DBC	1	10573		0.02	0.00	0		MONITOR	UNUSUAL		
DBC	1	10579		0.03	0.00	16		DBC/SQL	IDLE	00:00:01	00:00:01
DBC	1	10617		0.00	0.00	16		DBC/SQL	IDLE	00:00:01	00:00:01
DBC	1	10618		0.00	0.00	16		DBC/SQL	IDLE		00:00:01
DBC	1	10626		0.00	0.00	16		DBC/SQL	IDLE	00:00:01	00:00:01
DBCMANAGER	1	10589		0.00	0.00	0		MONITOR	IDLE		00:00:01

Note: You cannot abort or otherwise administer this internal session.

For more information, see "Handling Blocked Internal Requests That Slow the System" on page 407.

Query Data Storage and Protection

When query logging is invoked, query data is stored in multiple DBQL cache segments. If a user has query logging rules for a session, data is collected for each query that runs under that session. Collected data is retained in cache until a log row is committed.

Depending on the rules, a log row is committed and the cache is flushed when:

- An END QUERY LOGGING statement is received for a user who is currently logged on. The data may be incomplete.
- The cache is 80 percent full.
- A 10-minute interval elapses (for the SUMMARY option) and data is in the cache.
- The value for DBQLFlushRate is reached.

Because of its nature, there is no recovery mechanism for the cache in which DBQL row values are collected. Should a Teradata Database 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.

To clear all the DBQL caches, define an interval in seconds for the DBQLFlushRate field in DBS Control record. You can select a rate from 1 to 3600 seconds, however, the recommended rate and default is 600 seconds (10 minutes).

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 Teradata Database transaction and recovery journaling process. (For more information, see "Startup and Recovery" on page 292.)

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. (For details, see "AMP Clustering and Fallback" on page 231.)

Caution: Because dictionary tables are permanent, the contents of all logs remain until they are explicitly deleted. When you use DBQL, be sure to delete the rows from the logs on a regular basis to avoid running out of PERM space in DBC. For instructions, see "Maintaining the Logs" in the following section. (For instructions on checking available DBC space, see "Permanent Space Availability" on page 90.)

Maintaining the Logs

You can access the contents of the DBQL logs, but the following DBQL objects are protected:

- No user, including DBC and SystemFE, can access the DBQLAccessMacro or the DBQLRuleTbl or DBQLRuleCountTbl.
- No user can modify the DBQLAccessMacro or alter, drop, or update any of the DBQL tables.
- You cannot delete rows from the DBQLRuleTbl or the DBQLRuleCountTbl, because these are manipulated by BEGIN QUERY LOGGING and END QUERY LOGGING processing.

However, when logged on as user DBC or SystemFE, you can delete rows in the DBQL logs. This is necessary for controlling their size.

Note: Empty the DBQL logs **as often as possible**. Query logging consumes DBC PERM space and the rows remain in the DBQL tables until you remove them. Even though logging is not designed as a standard operation to be performed against all users all the time, the tables fill up very quickly. Especially if you are tracking target objects for a large group of users.

Caution: Do not off-load data during peak busy times as this may block ongoing queries.

When you are ready to use DBQL on your production queries, follow this procedure:

1 Create and maintain an executable BTEQ script file to submit (or keep a copy of) your final BEGIN QUERY LOGGING statements in case a Sysinit is ever required (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 the DBQLRuleTbl and rebuild your rules cache.

2 Set up a regular schedule to periodically perform the following maintenance (for a list of all the DBC logs that need to be maintained on a regular basis, see "Maintaining Data Dictionary Tables and Logs" on page 120):

a Regularly summarize the data you want to retain permanently for analysis and planning. You can save it in a user database, external files, applications, or BTEQ reports, and so forth. (Also, you can use Teradata Manager; see *Teradata Manager User Guide*.)

b Then submit the statement `DELETE * FROM DBQL` *tablename*`;` on each of the DBQL log tables (excluding DBQLRuleCountTbl and DBQLRuleTbl) to empty them and free up DBC PERM space.

You should schedule clean up activities during off-peak times. You may want to consider disabling DBQL logging before you perform clean up activities on the logs although this is recommended but not necessary. Otherwise, the delete process locks the DBQL table and if DBQL needs to flush a cache to the same table in order to continue logging queries, the whole system could experience a slow-down.

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:

- Regularly clean up DBQL log tables by off-loading them to history tables. This is especially true if you are also doing detail logging. (That is, logging steps, objects, SQL, or Explain). You should do this daily if not several times a day.
- Use value list compression on the columns of the copies of DBQL tables.
- Do not use fallback on your historical copies of DBQL tables. Using both NO FALLBACK and compression will save a lot of disk space.
- Log only the data you need. For example, if basic data 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 or account.

For further discussion on how to optimize DBQL logging, see *Performance Management*.

CHAPTER 14 Troubleshooting

This chapter suggests some tools for resolving problems and points to specific references for information on how to use these tools. Some of these tools can do things like help you find and analyze a problem. This chapter also describes how to handle when the system slows down or hangs.

Note: Access to administrator utilities is usually limited to privileges specifically granted in the database. Users of these utilities should log on using their Teradata 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 administrator utilities, see *Utilities*. For more information about the privileges of externally authenticated users, see *Security Administration*.

Tools for Troubleshooting

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 Teradata Database system developers. If the tools are described as for use by Teradata personnel only, do not use them unless instructed by Teradata Support Center.

For more information on the respective tool, see the chapter or book referenced. For a list of the utilities by suggested function (such as maintenance, troubleshooting, installation, configuration, or upgrade) see "Functional Listing of Utilities" in *Utilities*.

What to Troubleshoot or Administrate	Tool and Description	Reference
Administrative Workstation	The Administrative Workstation (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 (Visit www.info.teradata.com and search for the keyword "AWS".)
AMPs and AMP Worker Tasks	The ampload utility reports AMP bottlenecks due to unavailable free AWTs.	• "ampload Utility" in *Utilities*

What to Troubleshoot or Administrate	Tool and Description	Reference
AMPs and AMP Worker Tasks (continued)	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. awtmon is a front-end tool to the "puma -c" command and prints AWT inuse count information in a summary format. It works on MP-RAS, Microsoft Windows, and Linux. 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. Once hot AMPs are identified, run awtmon on those hot nodes using pcl or psh.	• "AWT Monitor (awtmon)" in *Utilities* • "Finding Busy AMP Worker Tasks" on page 393
Configuration changes	The PUT utility reboots nodes, sets up disk arrays, or changes the configuration of your system. PUT can perform configuration tasks such things as moving AMPs and PEs, changing clustering assignments, or partitioning Logical Unit Numbers (LUNs).	• *Parallel Upgrade Tool (PUT) for Microsoft Windows User Guide* • *Parallel Upgrade Tool (PUT) for UNIX MP-RAS and Linux User Guide*
CPU skew	Use PMON of Teradata Manager 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. You can also use Teradata DWM to control how much CPU a workload gets before it actually runs. Or, using PM/API, you could develop your own application to monitor for high CPU consumers.	• *Teradata Manager User Guide* • "EXPLAIN Request Modifier" in *SQL Reference: Data Manipulation Statements.* • *Teradata Dynamic Workload Manager User Guide*
Data corruption and integrity	Setting the statistical checksum through the DBS Control allows you to quickly detect data corruption. You can control the level of data sampling used for checksum verification. For more exhaustive checking, set the checksum level higher in the DBS Control. When the system detects corruption, the database restarts and does not allow access to the corrupt segment. If multiple accesses to a corrupt segment are attempted, FATAL AMPs results. After the system detects a corruption in the data, contact the Teradata Support Center.	See "Checksum Fields" in the "DBS Control Utility" chapter of *Utilities*

What to Troubleshoot or Administrate	Tool and Description	Reference
Data corruption and integrity (continued)	The SCANDISK command of the Ferret or Filer utility[a] allows Teradata Support Center personnel to check the integrity of the Teradata Database file system. This includes the master index, cylinder index, and data block structures (including structures associated with WAL logs). Previously halted SCANDISK operations can be restarted using a perl script. For more information, see "Filer Utility" in *Utilities*. **Note:** 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.	• "Ferret Utility" in *Utilities* • "Filer Utility" in *Utilities*
Defining vprocs and hosts	The Configuration utility[a] defines AMPs, PEs, and hosts and their interrelationships for a Teradata Database. This utility is used in conjunction with the Reconfiguration utility.	"Configuration Utility" *Utilities*
	The PUT utility also allows you to define and set up vprocs on the system.	• *Parallel Upgrade Tool (PUT) for Microsoft Windows User Guide* • *Parallel Upgrade Tool (PUT) for UNIX MP-RAS and Linux User Guide*
Disk space	The Ferret utility[a] allows field engineers and Teradata support people to monitor and control disk usage. The commands of this utility allow the system to combine free sectors on a cylinder, reconfigure contents on a disk leaving a specified percent of space free for future use, report which tables qualify for packing and display disk cylinder utilization and available free cylinders.	• *Utilities* • Chapter 3: "Space Considerations"
	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. 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 DBC Utility" in *Utilities*
	Update Space recalculates permanent, temporary, or spool spaces for one database or all databases.	"Update Space Utility" in *Utilities*
	The DBC.DiskSpace 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.	• "DiskSpace View" on page 100 • *Data Dictionary*
Global parameters	DBS Control displays and modifies the tunable global parameters in the DBS Control Record.	"DBS Control Utility" in *Utilities*

<table>
<tr><th>What to Troubleshoot or Administrate</th><th>Tool and Description</th><th>Reference</th></tr>
<tr><td rowspan="2">Gateway settings</td><td>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.
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.
<b>Note:</b> This utility is used by Teradata Database engineers to perform routine and special diagnostics.</td><td rowspan="2">• “Tools for Troubleshooting Client Connections” on page 392
• “Diagnosing Logon Encryption Errors” on page 398
• “Gateway Control (gtwcontrol)” in <i>Utilities</i>
• “Gateway Global (gtwglobal, xgtwglobal)” in <i>Utilities</i></td></tr>
<tr><td>Gateway Global 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.</td></tr>
<tr><td rowspan="2">Hangs or slowdowns</td><td>See “AMPs and AMP Worker Tasks” and “Resource usage.”
For more information on how to determine if the problem is a hang or slowdown, see “Telling the Difference Between a Hung and a Slow System” on page 406.</td><td>“Resource Check Tools (dbschk, nodecheck, syscheck)” in <i>Utilities</i></td></tr>
<tr><td>Resource Check Tools detects hangs or slowdowns as follows:
<table>
<tr><th>Tool</th><th>Description</th></tr>
<tr><td>dbschk</td><td>Checks if system is hung or congested. For instructions on how to use dbschk, see the man or pdehelp listed under References.
dbschk looks for the dbschkrc file during startup and if any options are specified in the file, those settings override the default options.</td></tr>
<tr><td>nodecheck</td><td>Displays local, node-level resources only. Provides summary data to syscheck for analysis.</td></tr>
<tr><td>syscheck</td><td>Analyzes relevant Teradata Database system data from the local node.
A text file called syscheckrc resides on each node. You can set it up to detect and reports any resource that falls below its predefined dangerous level.</td></tr>
</table></td><td>For information about the resource file for dbschk which controls the rates and options, see “Using the Resource Check Tools” on page 402.</td></tr>
</table>

What to Troubleshoot or Administrate	Tool and Description	Reference
Join Indexes when Restoring Tables	Teradata strongly recommends that you put all join indexes in databases strictly reserved for join indexes and never archive any of these databases. This will protect you from problems with invalid join index references when archiving. If you attempt to restore a table which is a party to a join index, you will receive a 5467 or 5468 error. First, drop all join indexes to which the table is a party. Then recreate them after the restore. There is no easy way to determine which join indexes are affected.	
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 ARC. Careful planning of what session 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.	See "Managing Locks" on page 215
	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 Utility" in *Utilities*
	The Locking Logger (dumplocklog) utility optionally logs queries that are queued due to locking contentions including non-ARC related global deadlocks. The log is useful for finding a session that regularly blocks others, particularly when you need to understand and resolve a deadlock. To use Locking Logger, you must first change the LockLogger field in the DBS Control record to TRUE. The default is FALSE.	• "Monitoring Lock Contentions with Locking Logger" on page 409 • "Locking Logger (dumplocklog) Utility" in *Utilities*
	Use the following command of the Checktable utility to find pending locks for FastLoad, MultiLoad, Restore, Reconfig, or Rebuild: `check all tables at level pendingop;` **Note:** Running CheckTable at the PENDINGOP level will not take up system resources and completes within minutes. Other Checktable levels which also report the same information do.	"CheckTable Utility" in *Utilities*
	Showlocks utility identifies and displays all active host utility (HUT) locks placed on databases and tables by ARC operations.	• *Teradata Archive/Recovery Utility Reference* • "Showlocks Utility" in *Utilities*
MPP environment	The xpsh utility is a parallel shell tool that provides a GUI front-end for performing various Teradata Database system-level tasks, such as debugging, analyzing, and monitoring an MPP environment. This utility is for MP-RAS.	"xpsh Utility" in *Utilities*

What to Troubleshoot or Administrate	Tool and Description	Reference
Network services	The tdnstat utility displays or clears statistics specific to Teradata Network Services.	• "tdnstat Utility" in *Utilities* • "Tools for Troubleshooting Client Connections" on page 392
Priority of jobs	Priority Scheduler assigns different priorities to different types of jobs. You can assign critical jobs more CPU and faster I/O than lower-priority jobs.	"Priority Scheduler" in *Utilities*
Queries	Manage your queries using Teradata DWM. You can: • 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. • Determine what Performance Group queries should run.	• "Managing Workloads with Teradata Dynamic Workload Manager" on page 318 • *Teradata Dynamic Workload Manager User Guide*
Resource usage	Use the Performance Monitor tool of Teradata Manager to access resource usage data. You can get information about which nodes or which vprocs are heavily using which resources. This can help you identify bottlenecks.	*Resource Usage Macros and Tables*
	If you are using MP-RAS, RSSMon utility provides real-time resource usage information per node and allows you to set the rate at which the system collects resource usage information through the Screen RSS option of the ctl utility.	"Screen RSS" of the ctl utility and "RSSMon Utility" in *Utilities*
	The Database Query Log (DBQL) view DBC.QryLog 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.	Chapter 13: "Tracking Processing Behavior with the Database Query Log (DBQL)"
	Teradata Database Workload Manager allows you to manage your resources by creating rules that filter out or throttle specific queries. You can also define workloads to specify a combination of resource usage limits and rules.	"Managing Workloads with Teradata Dynamic Workload Manager" on page 318.

Session information when logged onto multiple sessions	To find the Thread ID or Job ID of the session from which you are querying, first submit a SELECT SESSION;
	Then after that query returns the session number *n*, use that number in the following query:
	`sel logonsource`
	`from sessioninfox`
	`where sessionno = n;`
	
	`*** Query completed. One row found. One column`
	` returned.`
	`*** Total elapsed time was 1 second.`
	
	`LogonSource`
	`-------------------------------------------------------`
	`(TCP/IP) 09AB 127.0.0.1 DBC    8804  USER  BTEQ  01 LSS`
		"SessionInfo[V][X]" in *Data Dictionary*

What to Troubleshoot or Administrate	Tool and Description	Reference
Session information when logged onto multiple sessions (continued)	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.	"SessionInfo[V][X]" in *Data Dictionary*
	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 Utility" in *Utilities*
Slowdowns	See "Hangs or slowdowns" in this table.	
Spool space running out	Poorly written queries can sometimes consume spool space. Try using Teradata DWM 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 when using OLAP statistical functions 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. 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.	• "EXPLAIN Request Modifier" in *SQL Reference: Data Manipulation Statements* • "Statistics Collection" in *Teradata Manager User Guide* • "Ordered Analytical Functions" in *SQL Reference: Functions and Operators*
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 the Teradata Support Center.	• "ctl Utility" in *Utilities* for Windows and Linux • "xctl Utility" in *Utilities* for MP-RAS
	DBC.Software_Event_Log 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. For MP-RAS, the number is in the format of aaa-bbbbb-cc where bbbbb is a code you can look up in *Messages*.	• "Viewing the Software_Event_Log" on page 474 • *Data Dictionary* • *Messages*
	stune (UNIX) is a file you can use to help prevent UNIX panics (and thus system crashes) by modifying the LOTSFREE, DESFREE, and MINFREE parameters.	"Adjusting Available Free Memory (MP-RAS)" on page 398
	DUL utility saves or restores system dump tables onto tape.	"DUL Utility" in *Utilities*

What to Troubleshoot or Administrate	Tool and Description	Reference
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 Utility" in *Utilities*
Table	The Table Rebuild utility rebuilds tables the Teradata Database cannot recover automatically. Table Rebuild can rebuild: • The primary or fallback portion of a table • An entire table (both primary and fallback portions) • Only the primary or only the fallback portion of a table that reside on an AMP	• "Table Rebuild Utility" in *Utilities* • See "Table inconsistencies or corruption"
Table inconsistencies or corruption	CheckTable is a diagnostic tool that checks for: • Table and dictionary inconsistencies, such as differing table versions, ParentCount and ChildCount data, and partitioning definitions. • 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. To run CheckTable on an active production system (that is, a non-quiescent system), use CONCURRENT MODE. CONCURRENT MODE skips all locked tables and attempts to retry them after CheckTable finishes checking all of the specified tables that are not locked by other sessions. Options for CheckTable include the following: • The ERROR ONLY option causes CheckTable to only display skipped tables and tables with errors and warnings. This allows you to quickly identify and address problems found by CheckTable. • In PARALLEL mode, the TABLES = *n* option allows you to specify the number of tables checked in parallel. Checktable can also report if a table has been activated for online archive logging. You can use online archiving using the LOGGING ONLINE ARCHIVE ON statement if you want to concurrently archive and update transactions for the tables in the database at the same time. For more information, see *SQL Reference: Data Definition Statements* and *Teradata Archive/Recovery Utility Reference*.	"CheckTable Utility" in *Utilities*
	CHECKTABLEB is a non-interactive batch mode version of CheckTable that can be run through the cnsrun utility. CHECKTABLEB is identical to CheckTable except that it is meant to run from a script.	

What to Troubleshoot or Administrate	Tool and Description	Reference
TableSize view	If a SELECT statement on the DBC.TableSize view appears to hang, replace the definition of the view using the example code found in "TableSize View" on page 102.	"TableSize View" on page 102.
Teradata Performance Monitor	Use to find a down component, such as AMP, PE, or BYNET (via Start > Programs or through Teradata Manager).	• *Teradata Manager User Guide*
Transactions	The TPCCONS utility allows you to perform 2PC-related functions such as display host identifiers for all in-doubt transactions or list all coordinators and sessions with in-doubt transactions. You can also use it to resolve in-doubt transactions.	"TPCCONS Utility" in *Utilities*
UNIX hardware	The Abort Host utility aborts all outstanding transactions running on a failed host until the system restarts the host.	"Abort Host Utility" in *Utilities*
	streams log (in /var/adm/streams) on MP-RAS reports the condition of system hardware and UNIX activity.	Appendix C: "Error Logs"
UNIX system	The xcpustate utility or the xperfstate utility can monitor the state of CPU utilization in real time on MP-RAS systems. (xcpustate slightly impacts the system less than xperfstate.)	• "xperfstate Utility" in *Utilities* • "xpsh" utility for running xcpustate in xappl mode.
Vprocs and disks	From the Supervisor screen of the Database Window, view and administer the states of vprocs and disks.	*Graphical User Interfaces: Database Window and Teradata MultiTool*
	The Vproc Manager utility allows you to: • Initialize and boot a specific vproc • View or change vproc states • Initialize the vdisk associated with a specific vproc • Force a manual Teradata Database restart	• Chapter 10: "Stopping and Restarting the System" • "Vproc Manager Utility" in *Utilities*
	The Query Configuration utility reports the status of vprocs managed by node or for the entire Teradata Database system.	"Query Configuration (qryconfig)" in *Utilities*
vpacd utility	Improves performance of MP-RAS systems with several CPUs and high level of concurrency. The vpacd utility does this by binding tasks in each vproc to one or more CPUs. This improves performance by avoiding costly cache coherency updates over the system busses.	*Utilities*

a. Do not use the this utility unless instructed by Teradata Support Center.

Tools for Troubleshooting Client Connections

Channel Connection Tools

The following tools monitor session and TDP activity on channel-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 Teradata Database, and when the response parcel is received from Teradata Database.
- 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.
- MVS SMF - System Management Facility (SMF) is a mechanism that provides accounting and performance information on MVS, such as:
  - Statistical information about the processing activity of a PE recorded at shutdown.
  - Log-off session information, including the use of client and Teradata Database resources for a session.
  - Logon violations and security violations records.
  - Statistical data about the processing activity of the TDP, recorded at shutdown.

For more information on TDP, see *Teradata Director Program Reference*.

Network Connection Tools

The following tools monitor and control sessions originating from network-attached clients.

<table>
<tr><th>Tool</th><th>Description</th><th>Reference</th></tr>
<tr><td>Gateway Control</td><td>Utility that allows you to monitor network and session information.
<table>
<tr><th>IF you want to...</th><th>THEN use this command...</th></tr>
<tr><td>get network configuration information</td><td>DISPLAY NETWORK</td></tr>
<tr><td>see all sessions connected via the gateway</td><td>DISPLAY GTW</td></tr>
<tr><td>see status information for a selected gateway session</td><td>DISPLAY SESSION</td></tr>
<tr><td>force off one or more network sessions</td><td>KILL</td></tr>
</table></td><td>• "The Network Gateway" on page 445
• "Diagnosing Logon Encryption Errors" on page 398
• <i>Utilities</i></td></tr>
<tr><td>tdnstat</td><td>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.</td><td><i>Utilities</i></td></tr>
</table>

Finding Busy AMP Worker Tasks

When the system receives a query, it breaks the query down into one or more steps and sends them as a message to one or more AMPs. The steps try to get worker tasks from the pool of available AMP worker tasks (AWTs) to execute the work. After the step is complete, the AWT is returned to the pool. If all AWTs are busy at the time a new step arrives, then the message containing the step will wait in a queue until an AWT is free.

By default, there are a maximum of 80 AWTs per AMP[1]. 18 of the AWTs are permanently associated with internal activity, leaving a maximum of 62 AWTs available to service user requests. If a high number of AWTs are consistently in use, this could cause performance problems.

An example of when AMP Worker Tasks can become busy is if DBQL or Teradata DWM internal requests become blocked. (See "Handling Blocked Internal Requests That Slow the System" on page 407 for more information.)

There are three ways for you to detect busy AMP worker tasks. They are listed below.

Puma -c Command

The puma -c command is useful for finding hot AMPs or determining if the system if busy. On MP-RAS, run puma through /usr/ntos/bin/. On Windows, run the command at <*drive*:>/Program Files/NCR/Tdat/LPDE/bin.

If you run the following command:

```
puma -m | grep -v ' 0 '
```

and there are 20 or more messages per mailbox, the system is busy but is still processing work.

If you run the following command:

```
puma -c |grep -v ' 0 '
```

and the numbers from MSGWORKNEW and MSGWORKONE together are greater than 61, this probably means AWT congestion and that means the system is busy. If you notice just a few AMPs show higher AWT usage compared to the rest of the AMPs, these are hot AMPs and skewed queries are running.

You can use rallsh to run these commands on all nodes, but this may result in an enormous amount of output. To get a condensed summary report, use the awtmon utility.

[1] An internal DBS Control Record field called MaxAMPWorkerTasks is set to the default of 80. However, do *not* change this value unless you have carefully consulted with Teradata Support Center. Increasing the maximum number of AWTs per can cause severe resource contention such as running out of memory. Being out of AMP worker tasks is not a bad thing in and of itself. Teradata designed AWTs to control and manage workflow.

DBC.ResUsageSAWT Table

The system collects resource usage data for the DBC.ResUsageSAWT table if logging is enabled. The table holds information about the mailbox for each AMP, whether or not the system is in flow control, and the maximum number of AWTs for a vproc on a node. For more information, see *Resource Usage Macros and Tables*.

ampload Utility

The ampload utility allows you to view how many messages are waiting on which AMPs on which nodes to determine busy or “hot” AMPs. In particular, the ampload utility reports the vproc number, node id, number of messages waiting on each AMP (the message queue length), and the number of AWTs available per AMP.

AWT Monitor (awtmon) Utility

AWT Monitor (awtmon) displays the AWT in-use count information, just as the puma -c command, in a summary format.

By default awtmon displays the AWT in-use count for a local node. However, the -s option gathers and summarizes system-wide AWT in-use information. On a large system, the -t option helps reduce the overall output size. Typically you should use -s and -t options together on a large system to filter out the output size.

You can take a few snapshots of awtmon to locate any hot amps on each node in the system. Once hot amps are identified, run awtmon on those nodes using pcl or psh. So for example, if you are on an MP-RAS system and a snapshot revealed that byn001-5 turned out to be a hot-node in the system. Run awtmon on byn001-5 node to monitor it as follows:

```
$ pcl -nodes byn001-5 -sh awtmon
```

or

```
$ psh
  psh.0> sel byn001-5              ; select the node in interest
  1 node selected
  psh.1> awtmon                    ; run awtmon
       .
       .
       .
  psh.x> quit
```

For output examples and descriptions of available options, see *Utilities*.

Differences Between ampload and awtmon Utilities

The differences between ampload and awtmon are described in the following table.

awtmon	ampload utility
awtmon prints the break down of the AWT INUSE count by worktype, that is, WORKNEW, WORKONE, etc. Breakdown of worktype for active AWTs is useful for troubleshooting performance problems, and more specifically, finding hot-amps per node.	ampload only prints AWT INUSE count.
awtmon prints local nodes AWT INUSE count information by default and [-s] option for system-wide information via pcl.	When invoked from a command prompt window, ampload displays only local node info. When invoked from the CNS/DBW supervisor window, ampload prints only system-wide info.
awtmon provides [t [*n*]] options for a loop mode	ampload does not provide options for a loop mode. You must manually submit the ampload command repeatedly if you wish to collect several reports for comparison.
awtmon provides [-t *count*] option to filter out and reduce the output to report any vproc where the AWT in-use count is less than a number you specify.	ampload always reports the vproc number, node ID number, message count, and AWT availability count for each vproc across the entire system.
awtmon attempts to reduce the number of outputs by default. This helps reduce the output on a large MPP system. Also, no line is printed if a vproc's AWT in-use count is ZERO.	ampload prints a line for each vproc even when AWT in-use count is ZERO.
awtmon does not report the number of messages waiting on the AMP.	ampload allows you to see how many messages are waiting on the AMP.

Resolving Lack of Disk Space

This section provides suggestions for analyzing and solving lack of disk space.

IF you...	AND...	THEN try resolving the problem by ...
are getting messages that DBC is out of space even though the system still has a lot of unused space	when you add even more space and clean up the system, you still get an error	checking for table skewing in the system. In particular, check DBC.AccessLog table to see if it is skewed and if you need to clean it out. For more information, see "Table Skewing" on page 396.

IF you...	AND...	THEN try resolving the problem by ...

tried manually adding permanent space to a user	you get the following error message:
	`"3541: The request to assign new PERMANENT space is invalid."`
		making sure that there is enough space in the parent.
		Allocate space by submitting something like:
		`CREATE TEMP FROM DBA_RESERVE AS PERM = xxxx;`
		`GIVE TEMP TO target;`
		`DROP DATABASE TEMP;`
		or use the Move Space button in Teradata Administrator (formerly known as WinDDI).

use DBC.DiskSpace or Teradata Manager to invoke the error log analyzer	the results show that the system is low on available permanent space	• Running PACKDISK. • Setting spool limits on users. • Running DEFRAGMENT. • Cleaning out system logs. • Considering to add disks.
• run SHOWSPACE (Ferret) • run SHOWFSP (Ferret) to find good candidates for packing.	the system has adequate disk space, but is out of free cylinders.	• Running PACKDISK on the tables reported by ShowFSP; specify an increased free space amount. • Using DBS Control utility to change FreeSpacePercent globally if necessary. Note that you will have to do a tpa restart. • Using ALTER TABLE to change the free space percent for one or more growth tables if necessary.
ran Teradata Manager to invoke the error log analyzer	the results show that a very large spool is being requested	• Using Query Session or Performance Monitor to find the job requiring the space, then cancel the job with one of: • Performance Monitor (Windows and Teradata Manager) • TDP LOGOFF command • Gateway KILL command • Conferring with the applications programmer to correct the structure of the SQL query.

Table Skewing

When you allocate space to a database, the space is evenly allocated to all AMPs. For example, a four AMP system will get 1GB per AMP if you allocate a database of 4 GB. The system will report a problem of running out of space if the first AMP reaches its 1GB limit.

As a worst-case scenario of this example system, assume you insert data into a table and find that all records hash to a single AMP. In this case, you will only be able to insert 1GB worth of data (filling up one AMP) even though the system will still report 3 GB of free space from the other 3 AMPs that are empty.

If you run the following query:

```
SELECT databasename, vproc, sum (currentperm) as cp, sum (maxperm) as mp,
 mp - cp as free
FROM DBC.allspace
GROUP BY 1,2
ORDER BY 1,2
WHERE tablename = 'all';
```

ideally, the values in the last column ("free") will be about the same (more or less within about 5 percent) for any given database. If this is not the case, the tables are skewed.

You can look at individual tables within a database to see which one is skewing the most. Use a query similar to the following:

```
SELECT tablename, vproc, currentperm as cp
FROM DBC.allspace
ORDER BY 1,2
WHERE databasename = 'dbc';
```

Again, the tables should have similar amounts of space on each vproc. Especially those with large numbers of rows.

The following tables should be carefully managed because they are prone to skewing.

- Journal tables. Journal space (spool and other transient storage) typically comes 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.AccessLog table. Teradata Database generates at least one entry every time a user defined for logging attempts to access an object. This table can grow quite large and consume a lot of disk space.

Every table in the system, however, is prone to becoming skewed if the index is poorly designed or if statistics have not been collected and the Optimizer is using extremely 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.

Adjusting Available Free Memory (MP-RAS)

To protect against MP-RAS panics and prevent wasting free memory, you can increase the values of the page parameters in /etc/conf/cf.d/stune. Higher pages enable the OS to start paging sooner and thus free up memory sooner. The **stune** parameters and recommended values are listed in the following table.

Change this parameter...	From default (pages) of...	To (pages)...
LOTSFREE	512	8192
DESFREE	256	4096
MINFREE	128	2048

For more information on other settings associated with memory on MP-RAS, see the following section titled RedistBufSize Performance Field and Adjusting for Low Available Free Memory in *Performance Management*.

Diagnosing Logon Errors

There are a number of reasons that might cause logon failures. Errors could range from incorrect case usage for passwords to incorrect settings in security mechanisms. Two possible problems with logging in, discussed in the following section, could result from mismatched settings for authentication or encryption settings.

Note: This section does *not* contain an exhaustive list of how to diagnose logon errors. For example, it does not discuss errors that could result from the LDAP method and the directory server you have chosen. For a full discussion on logon errors, see *Security Administration*.

Diagnosing Authentication Errors

You may not be able to properly log on to Teradata Database if the settings for external authentication in the DBS Control utility and Gateway Control utility conflict. For an explanation of the Gateway error messages, see *Messages*.

The DBS Control utility affects all logons to the system while the gtwcontrol utility only affects logons through a particular Teradata gateway. For more information on the utilities, see *Utilities*. For more information about the logon settings, see *Security Administration*.

Diagnosing Logon Encryption Errors

If you are administering a Windows site that supports SSO external authentication, a high incidence of rejected logons might indicate a mismatch of the settings of the external authentication field in the Gateway Control GDO and the DBS Control GDO.

For detailed instructions, see "Single Sign-On" on page 188.

Mismatch of External Authentication Values in DBS and Gateway GDOs

Errors occur if users submit logons with passwords when both the DBS and the Gateway GDOs are set to external authentication as ON (or when DBS Control record is set to ON and Gateway Control record is set to ONLY. See *Graphical User Interfaces: Database Window and Teradata MultiTool* for more information on SET SSO and SET EXTAUTH). Both DBS and Gateway GDOs must be set to the same external authentication values and your clients should be aware of the required procedure.

Note: A DBS Control record setting of ONLY or OFF overrides any gateway setting.

- Reset the ExternalAuthentication value in the DBS Control record, use the Supervisor command: set extauth [on/off/only]
- To reset the external authentication setting in the Gateway to a matching value, use the gtwcontrol command with the -a option.

  ```
  gtwcontrol -a on/off/only
  ```

  Include the -g option if you use client groups:

  ```
  gtwcontrol -g  hostid -a on/off/only
  ```

  where *hostid* is the unique identifier you assigned to a client group. (For details, see “Logon Controls” in *Security Administration*.)

Nonunique Usernames

For systems using NTLM, KRB5, and LDAP where LDAP is Active Directory and the sever is running on Windows, another setting that might cause external authentication errors is the Append Domain Name field in the Gateway Control GDO. Attempts to log on fail when:

- A username cannot be recognized as unique across all domains in the network. (To guarantee uniqueness, you can append your usernames with domain names; see “Password Security for Users” on page 131.)
- The form of username being submitted does not agree with the form defined as valid by the Append Domain Name field. Two forms are possible, but only one can be in force at a time.

The choices for forms of username are as follows:

IF your site ...	THEN the value of Append Domain Name in the Gateway GDO should be ...
does not use domain names (userID is *username*)	no.
uses domain names (user ID is "*username@domainname*")	yes. However, this value is deprecated in Teradata Database 12.0 and the Append Domain Name value will be permanently set to no in future releases.

To enable one form or the other, use the gtwcontrol command as follows:

IF you want to ...	THEN use gtwcontrol with ...
see the current settings of the Gateway Control GDO	the -d option, as follows: `gtwcontrol -d`
toggle the value of the Append Domain Name field	the -F option, as follows: `gtwcontrol -F` **Note:** The -F option is deprecated because it will be removed in future releases. For more information, see *Utilities*.

Preventing Slowdown or Hang Events

Detecting potential problems ahead of time includes monitoring transaction processing and looking at how often a resource is in use during a given period for patterns. You should ask the following questions:

- Is one query consistently blocking others?
- Are there many transaction deadlocks during peak workloads?
- Are all AMPs working equally hard?
- What are the disk loads and I/O counts?
- Is traffic on the BYNET moving normally?

The following section discusses some ways to minimize the occurrence of impacted performance. For more detailed discussion, see *Performance Management*.

Controlling Session Elements

All database systems reach saturation from time to time, particularly in ad-hoc query environments. However, on Teradata Database you can control session elements such as user spool space and job entry, and thus you can minimize how often end-users might saturate the database capabilities.

Teradata Database provides several tools with which you can control these elements. The most commonly used are introduced in the following table.

IF you want to …	THEN use one or more of these tools …	AND for further details, see …
control logon access	• Logon controls such as passwords, settings in profiles, or assignments to specific roles • Host group IDs, to authorize logons from specific client platforms with GRANT/REVOKE LOGON… *host_groupid* • Teradata DWM, to control access to objects as well as active sessions by user, account, performance group, and users within performance group	• Chapter 7: "Controlling and Tracking Access to the Database" • "Logon Controls" in *Security Administration* • "Managing Workloads with Teradata Dynamic Workload Manager" on page 318 • *Teradata Dynamic Workload Manager User Guide* • "GRANT LOGON" in *SQL Reference: Data Definition Statements* • *Teradata Director Program Reference*
control object access	• User spool space, to limit response sizes • User, role, or object privileges with GRANT/ REVOKE • Implement operations so that users access portions of data through views, macros, and stored procedures • Teradata DWM, to: • Control access to database objects • Limit access based on query type • Limit the number of active queries by user, account, performance group, and users with a performance group	• Chapter 6: "Controlling and Tracking Privileges" • "GRANT" in *SQL Reference: Data Manipulation Statements* • *Teradata Director Program Reference*
set up automatic job scheduling	• Priority Scheduler Administrator (PSA) to schedule priority of account access to resources such as CPU and memory • Teradata DWM, based on concurrent sessions, query type, account priority, quantity of response rows, or workload flow	• "Managing Resources with Priority Scheduler" on page 312 • "Managing Workloads with Teradata Dynamic Workload Manager" on page 318 • "Priority Scheduler" in *Utilities* • *Teradata Query Scheduler Administrator Guide*
determine if an upgrade or expansion of your Teradata Database would help avoid future resource saturation	• Baseline profiling comparisons • Resource Check Tools • ResUsage reports	• "Solving Bottlenecks by Expanding Your Teradata Database Configuration" on page 412 • "Using the Resource Check Tools" on page 402 • *Resource Usage Macros and Tables* • *Performance Management*

Ensuring Node Parallel Efficiency

Node parallel efficiency is not a matter of heaviness of workload; it is a measure of how evenly the workload is shared among the nodes. The more evenly the nodes are shared, the higher the parallel efficiency.

Node parallel efficiency is calculated by dividing average node utilization by maximum node utilization. The result indicates your node workload distribution.

IF node parallel efficiency ...	THEN ...
is nearly 100 percent	the better the nodes are working together.
falls significantly below 100 percent	in that time period, one or a few nodes are working harder than the others.
falls below 60 percent more often than a couple of sampling periods	your installation is not taking advantage of the parallel architecture.

Possible causes of poor parallel node efficiency include:

- Down node
- Non-Teradata application running on a TPA node
- Co-existence system with non-optimal balance
- Different number of AMPs on nodes of the same type
- On one or more AMPs down and the system running in fallback mode

AMPs always run in parallel, but the way data rows 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 or nulls, the majority of rows might hash to the same AMP.

To report the AMP bottleneck due to unavailable free AWTs, use the ampload utility. You can access ampload from the Supervisor window in Database Windows or the Teradata Command Prompt. For more information, see *Utilities*.

If your disk loads are poorly balanced, consider ways to correct the situation. For example:

- Perhaps queries or views against a column with zero or null values could use "WHERE NOT NULL" or "NOT= 0" qualifiers.
- If the cause is a nonunique primary index, consider redefining the index, especially for a very large table, to achieve a higher percentage of uniqueness.

Using the Resource Check Tools

Although the set of utilities in Resource Check Tools (RCT) is useful for identifying a slowdown or hang, you also can use them periodically to expose a potential problem before it impacts production.

The process is as follows:

1. After Teradata Database is installed, determine a reasonable response interval for the database. Use this as the parameter to dbschk.
2. Using the response interval you determined in step 1, run dbschk as a background task to continually monitor the response.

   Note: Run dbschk only when DBS logons are enabled (system status is: `*Logons are Enabled*`).

   By default, dbschk is turned off.

To turn dbschk...	Use the following command...
ON	`dbschk -power 1`
OFF	`dbschk -power 0`

3. Look at your site-specific copy of the syscheckrc file to see whether a value is set at a dangerous low for a resource, such as UNIX free memory, free swap space, or AMP worker tasks. For example, the node-only section of syscheckrc would include the values listed in the following table.

Resource value	Comment
Free memory	Below 1000 could be a warning level.
PDE Msg Daemon Queue Length	Above 100 msg counts implies congested state.
BNS Blocked Queue Length	Above 200 msg implies congested state.
Available AMP worker task	Below 2 implies available AWTs are running low.
BNS Msg Reject %	Above 80 percent implies congested state.

   Note: Congested means that the node (local or system-wide) is very busy or heavily loaded.

   Along with time-out rates, collection rate, job options, debug options, and delay options, this resource file also contains the settings for and trigger option. In particular, the trigger option allows you to run a script for collecting system data when system response time is above the “Expected Time Delay.”

   For MP-RAS and Linux, specify the trigger as:

   ```
   TRIGGER=/home/username/scriptname
   ```

   For Windows, specify the trigger as:

   ```
   TRIGGER=c:\temp\scriptname
   ```

   Be sure to define the trigger carefully so that it does not cause the system to hang or restart.

4. Create a site-specific file by doing one of the following:
   - Either copy the default file to a location as indicated below

- Or use the nodecheck utility with the following options:
  - First use the -D option (to redirect output and create an rscfilename that you can customize)
  - Then use the -r rscfilename option to read the created file

A variation of syscheckrc resides on each node, as follows:

File Description	Location
Default syscheckrc: • Should not be modified • Is the resource file for nodecheck and syscheck.	• On Windows: Program Files\NCR\TDAT\LPDE\etc • On MP-RAS: /usr/ntos/etc • On Linux: /usr/pde/etc
Your site-specific copy can be modified for your site to specify: • WARN and ALERT levels for each system resource • Sample number and sleep time.	• On Windows: Program Files\NCR\TDAT\tdConfig • On MP-RAS: /ntos • On Linux: /opt/tdat/tdconfig

5 If you see a LOGEVENT generated by dbschk in the stream log, which indicates that the response from Teradata Database exceeded the interval specified as reasonable, you should:

a Consult with daily operations to find out why the slowdown or hang occurred.

b If operations cannot explain the event, go to step 5.

6 Run the syscheck utility to see if any of the resources defined in syscheckrc are at the WARN level.

For More Information

For more information, see:

- "Resource Check Tools" in *Utilities*
- At the MP-RAS or Linux command prompt submit the following:
  - *`man dbschk`*
  - *`man syscheck`*
- At the Teradata command prompt submit the following:
  - *`pdehelp dbschk`*
  - *`pdehelp syscheck`*

Troubleshooting a Slow or Hung Job

If Teradata Database slows down or hangs unexpectedly, check:

- For down hardware.
- If the system is busy as a result of resource bottlenecks, spool file sizes, or data distribution across the AMPs.
- If there are enough AWTs by using the ampload utility or the awtmon utility.
- If there is a blocked job because of lock contention. For example, you may have to release locks manually if an ARC operation still remains.
- If the system is having problems with saving dumps which can cause performance problems.

Teradata Manager should be able to help you identify if these problems exist. However, you can also use the utilities and tools in the following table to help determine the problem.

Tool	Comment
ampload utility	Determine which AWTs are busy and reallocate your AWT assignments as necessary.
AWS console	Check the status of all hardware components.
BTEQ	Log onto Teradata Database and try to select time and date.
CheckTable LEVEL 3 command (if you use PPIs)	Make sure rows are properly located. If errors are reported, see "Solving PPI and RI Validation Errors" on page 411.
Lock Display command (Locking Logger utility)	Investigate whether queries are blocked by a long-running job holding a non-shareable lock.
qrysessn utility	Run qrysessn from the supervisor screen. If you find any blocked sessions, the utility will show the database and tablename on which it is blocked. You may have a deadlock condition.
ResUsage reports	Determine congested resources such as events that are I/O or CPU intensive, AMPs with a higher number of last-done events, etc. **Note:** In general, use Resource Usage through the Performance Monitor facility on a regular basis. Resource Usage is able to organize the data in a useful way, while Performance Monitor can provide real-time data analysis.
ShowLocks	Run Showlocks from the Supervisor screen to check for Host Utility Locks. Make sure an archive job did not leave active HUT locks on data tables. Note that this tool will not show transaction locks. If you find a host utility lock that is causing the problem, you can release the lock through BTEQ with the following command: `release lock dbname, override;`
syscheck	See if any kernel attributes are reaching a saturation level.

Tool	Comment
Software_Event_Log view	Check for messages from hardware errors, also for error 2631 records indicating repeated locking contentions.
Teradata Database log (on UNIX, /var/adm/streams)	Look for messages indicating many mini-cylpacks, deadlocks, memory paging, and so on. Check the streams logs on all nodes to see what transpired before the hang condition. You should look for: • mini-cylpacks (cylpacking) which could indicate full disks. • many "BNS PagePool" messages which could indicate that the system is running out of UNIX memory. • 2631 deadlock time-outs could indicate a deadlocking problem Any error messages should be checked for any possible problem.

• Update Space utility
• Update DBC utility
• DBC.DiskSpace view
	To rule out lack of available disk space, perform this procedure:
	1 Use the Update Space utility (for all types of space and all databases) to update current user space values:
	`UPDATE ALL SPACE FOR ALL DATABASE;`
	2 Use the Update DBC utility to update the values for system user DBC.
	`UPDATE DBC;`
	3 Query the DBC.DiskSpace view to find currently available space with this statement:
	`SELECT MAX(CurrentPerm), SUM(CurrentPerm)FROM DBC.DiskSpace ;`
	(For help, see "DiskSpace View" on page 100.)

Vproc Manager utility	Determine the current state of each AMP and of Teradata Database. For example, is Teradata Database in debugger mode?
xcpustate or xperfstate (UNIX)	To graphically display the current CPU state, use the xcpustate or xperfstate utility. (xcpustate slightly impacts the system less than xperfstate.) Run xcpustate in xappl mode of the xpsh utility. (See "xpsh Utility" in *Utilities* for more information.)
puma (/usr/ntos/bin/)	Use the puma -p command from the command prompt to determine the number and process id of tasks, mailboxes, and monitors in use (For example, /usr/ntos/bin/puma -p). Look for a node or process that is different than the others. Kill processes or end tasks that are hung.

Telling the Difference Between a Hung and a Slow System

At times, an extremely busy system can be so saturated with work that it may seem as though it has hung. Use xcpustate on MP-RAS or the Teradata Manager dashboard to determine CPU utilization.

IF CPU is...	AND ...	THEN the system is...
idle	• you cannot log on either to the database itself • or you cannot start a console window (such as Database Window Supervisor Window)	hung. In most cases, some process is unable to complete for whatever reason but may be tying up some resource and other processes are waiting behind it. You may have to reset the system.
busy	AMP worker tasks are busy but the system is still processing messages	busy. Look for queries or rollbacks you can abort. Also look for lock contention.

Use the "Puma -c Command" to determine if AWT counts are changing. If the counts are changing, this means the system has not hung.

Also see if running the puma -m command a few times shows a changing number of messages. Changing numbers means the system is not hung and is still processing messages.

Handling Blocked Internal Requests That Slow the System

DBQL and Teradata DWM use express requests to do internal work on DBQL and Teradata DWM log tables. If you are performing work on the same log tables, such as deleting rows, this might block DBQL or Teradata DWM 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 or Teradata DWM logging.

Session Information (SI), Performance Monitor (PMON), and the Monitor sessions option in Teradata Manager all report these blocked requests with an artificial internal session record with hostid of 0 and session number 2. This way, you can address the sessions that are blocking the internal work.

Note: You cannot abort or otherwise do anything to this internal session record. It exists only to show that internal DBQL and/or Teradata DWM requests are blocked.

If you are logging DBQL or Teradata DWM data, and you notice slowing, do the following:

1. Submit the MONITOR SESSION command of Teradata Manager. (You must use Teradata Manager Version 7.0.0.1 or later). If there are blocked internal requests, you will see this artificial internal session that represents the top three DBQL/Teradata DWM blockers.
2. If there are more than three blockers, fix those first three and re-display the monitor session information to display the following next three blockers. You can release a session from the delay queue or abort its query.
3. Continue fixing the blockers and re-displaying the monitor session information so that it will refresh and display the next blockers. Repeat this until the artificial internal session record no longer appears.

If DBQL or Teradata DWM internal requests remain blocked, the system deadlock detection may abort them. If the system aborts these internal sessions, the logging data will be lost.

Note: For DBQL, 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 or Teradata DWM internal requests reported by the internal session, consider scheduling your queries (such as table maintenance) on DBQL or Teradata DWM tables when there is no logging activity. Or use locks that hold the logging tables for a minimal amount of time.

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 to assure data integrity. However, rollbacks could involve a very large number of rows if the transaction involved a very large number of rows. Rollbacks could also keep many locks on the affected tables until the rollback completes, which could be a very long time.

If you decide that restoring specific tables would be quicker than waiting for the rollback to complete, cancel the rollback using the Recovery Manager (RcvManager) utility.

Note: You must first list rollback tables using the LIST ROLLBACK TABLES command and then cancel rollback from that list within the same Recovery Manager session.

For more information, see “Canceling Rollbacks” on page 274. Also, see *Utilities* for more information on the Recovery Manager utility.

Noticing a Slow Rollback of an ALTER TABLE Statement

Teradata ensures that all transactions are atomic meaning that a transaction must succeed or fail in its entirety. The system does not allow partial results. In most cases, if a transaction fails, Teradata rolls back any of the previous actions of a transaction. This requires that the system logs the "before" state of every action so that the action can be undone later if necessary. 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, however, this logging would be extremely expensive compared to the cost of the action itself: dropping a large table for example. If the system did not have to maybe roll the table back later, the system could just release all of the data of the table entirely, deallocating whole cylinders with a single action. In this case, Teradata would defer the action. It is deferred in the sense that 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 because 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 a roll back of ALTER TABLE is taking a very long time but you cannot abort it, the system is deferring the action and will need to complete.

Monitoring Lock Contentions with Locking Logger

The Locking Logger utility, also known as dumplocklog, allows you to save information about local and global transaction lock contentions.

The utility provides a tool for creating a table that stores data extracted from the buffers. (You cannot access system buffers directly.) Query this table to identify the session or sessions causing a locking queue delay.

When a transaction is blocked due to a lock delay, the AMP writes an entry in its lock log buffer. The entry includes the level of lock being held and the transaction, session, and locked-object identifiers of the involved request.

If you only want to log blocked lock requests that have been waiting more than a specific number of seconds that you have set as the limit, use the LockLogger Delay Filter and LockLogger Delay Filter Time DBS Control record fields. Locking logger checks blocking delays against these fields to determine whether the delayed lock request should be logged. For more information on these fields, see “DBS Control (dbscontrol)” in *Utilities*.

If the same query or group of queries appears consistently in the lock log table, use DBQL and Teradata Visual Explain to analyze, determine, and eliminate the cause. Be sure you have collected current statistics before critiquing the query structure or processing results. You can use the Statistics Wizard to determine what columns to collect statistics on. If you have enabled QCF, you can use either or both forms of the COLLECT STATISTICS statement. You can use the Index Wizard to analyze the efficiency of indexed access.

For instructions on how to enable the utility, see “Locking Logger (dumplocklog)” in *Utilities*.

Solving Lock, Partition Evaluation, or Resource Problems

The following table lists some tools that may help you can resolve some problems quickly.

<table>
<tr><th>Issue</th><th>Action</th></tr>
<tr><td>AMP and Disk CPU usage</td><td>If you have enabled ResUsage reporting, use ResUsage data to determine whether:
<ul>
<li>Disk I/O counts seem too high. You may be able to reduce I/Os by modifying the Cylinder Read default values. For details, see “Managing I/O with Cylinder Read” on page 309.</li>
<li>Disk and AMP CPU usage values are the same or different:</li>
</ul>
<table>
<tr><th>IF disk and AMP CPU values…</th><th>THEN …</th></tr>
<tr><td>are the same</td><td><ul><li>There may be a problem with the client connection. See “Tools for Troubleshooting Client Connections” on page 392.</li><li>You may need to add AMPs, PEs, disks, or nodes for more capacity. See “Solving Bottlenecks by Expanding Your Teradata Database Configuration” on page 412.</li></ul></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Issue</th><th>Action</th></tr>
<tr><td>AMP and Disk CPU usage (continued)</td><td><table>
<tr><th>IF disk & AMP CPU values…</th><th>THEN …</th></tr>
<tr><td>are different</td><td>resource utilization is skewed. There may be a problem in one of the AMPs or PEs, or uneven data distribution. Check the hardware or applications as follows:
• For uneven data distribution, discuss improving primary index uniqueness with operations or application designers.
• For an AMP or PE problem, use Performance Monitor or the Vproc Manager utility to fix it.</td></tr>
</table></td></tr>
<tr><td>Deadlock</td><td>If you run multi-tier applications with many network users logged on under the same userID, you can find the originator of a problem request by using one of the following to find the session:
• Query Session Utility
• Teradata Performance Monitor
• DBC.SessionInfo view
Query the LogonSource column of SessionInfo to obtain the TDPID, user name, and executable name of the session. For instance, in the example under “DBC.SessionInfo View” on page 200, the first network session logged on via TDP IETTST by user ADMIN using BTEQ.
When you have identified the session and user, you can choose to do one of the following:
• Dynamically lower the priority of the heavy user with one of:
  • SQL: SET SESSION ACCOUNT=’*prioritystring*’ FOR [*sessionID/requestID*]
  • Performance Monitor: MODIFY USER ACCOUNT facility
  • PM/API: SET SESSION ACCOUNT
• Abort the heavy user manually with one of the following:
  • Performance Monitor ABORT SESSION facility
  • TDP LOGOFF command
  • Gateway KILL command</td></tr>
<tr><td>HUT locks</td><td>Run ShowLocks to find any outstanding host utility (HUT) locks.<table>
<tr><th>IF …</th><th>THEN immediately …</th></tr>
<tr><td>locks are listed</td><td>on the client or a NetVault server, start an ARC session and submit the RELEASE LOCK command.</td></tr>
<tr><td>no locks are listed</td><td>run the Lock Display utility or, if you enabled the locklogger option, the dumplocklog command of the Locking Logger utility to check for transaction lock contentions.
• If the utility shows a bottleneck caused by an active session, see “Deadlock” on page 410.
• If no active locking queues or deadlocks are reported, review other issues.</td></tr>
</table></td></tr>
</table>

Issue	Action
Transaction rollbacks with partitioned tables	If transaction rollbacks are occurring because a partitioning expression is resulting in evaluation errors, do one of the following: • Change the partitioning expression • Delete the rows causing the problem • Remove partitioning from the table • Drop the table For more information, see "Solving PPI and RI Validation Errors" in the following section.

Solving PPI and RI Validation Errors

You can use the following procedures and tools to detect and correct errors in tables that use partitioning and referencing. (For details, see "Partitioned and Nonpartitioned Primary Indexes" in *Database Design* and "Using Referential Integrity" on page 240.)

IF you want to ...	THEN ...
correct a partitioning expression that is causing transaction rollbacks due to an evaluation error (such as divide by zero)	do one of the following: • Change the partitioning expression • Delete the rows causing the problem • 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 a PPI	use the ALTER TABLE … REVALIDATE PRIMARY INDEX statement.
for a table with a PPI: • Regenerate table headers • Re-evaluate partition expressions • Recalculate row hash values • Move rows to proper AMPs and partitions • Update any SI, JI, and HI defined for the table	use the ALTER TABLE … REVALIDATE PRIMARY INDEX null_partition_handler WITH DELETE/ INSERT[INTO] statement. • WITH DELETE deletes any rows with a partition number outside the range 1-65535. • WITH INSERT [INTO] deletes any rows with a partition number outside the valid range and inserts them into save_table. **Note:** REVALIDATE changes the table version.
reset the state of RI tables after an ARC RESTORE operation	run the ARC utility REVALIDATE REFERENCES FOR command.

IF you want to ...	THEN ...

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:
	`SELECT DISTINCT childtablename.*`
	`FROM childtablename,parenttablename WHERE`
	`childtablename.fkcol NOT IN (SELECT pkcol`
	`FROM parenttablename)`
	`AND childtablename.fkcol IS NOT NULL;`
	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.)

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 in order to maintain the integrity of your database.

Solving Bottlenecks by Expanding Your Teradata Database Configuration

System saturation and bottleneck identification are interrelated. When your Teradata Database is saturated, the bottleneck is usually some key resource, such as a CPU or disk. Use the information obtained from performance monitoring, resource usage, and query capture and process tracking tools to find the cause of repeated bottlenecks.

If a resource has been a bottleneck consistently during peak utilization periods and you have determined that your database design, data modeling, and query structures are efficient, consider expanding your Teradata Database configuration to improve performance.

Expansion involves adding any combination of disk arrays, memory, vprocs, or nodes (with BYNETs), and then running the Parallel Upgrade Tool (PUT) and Configuration and Reconfiguration utilities.

The Reconfiguration Estimator utility can provide an estimate of the duration of outage based on parameters you supply interactively. (For an overview of the utilities and a summary of procedures, see “Viewing or Modifying the Teradata Database Configuration” on page 431.)

Note: Make sure your applications are scalable and can take best advantage of the expansion. For guidelines, see “Scaling Your Applications” on page 415.

Determining Resource Needs

When planning expansion, you need to determine whether your configuration needs more memory, a more powerful or additional processors, more nodes, more disks, more disk array controllers, or additional options for OLTP environments.

To do this, analyze the performance information on AMPs and PEs, including:

- System usage
- Resource usage across AMPs and PEs

- The amount of disk I/O, BYNET traffic, and client I/O that is occurring
- Whether congestion or excessive swapping is a problem on any AMP or PE

Often you can satisfy the need for increased capacity or throughput by adding disks, memory, or vprocs. If that does not suffice, you can add nodes. In particular, you must add nodes when your system is CPU bound.

Adding Disk Arrays

When you add disk arrays, you increase the capacity for disk I/O.

This is helpful both for a DSS environment with a growing database, and for an OLTP environment with increasing concurrent disk I/O.

To determine if the system needs more storage, look at the ResUsageSvpr table for unusual disk activity, such as frequent:

- Mini-cylpacks
- Defrags
- Packdisks

You may need to add more storage capacity to existing nodes when:

- Excessive disk activity is impacting performance
- Application changes require additional spool space
- Database growth requires additional storage

When you add disks, you run utilities to slice and assign them automatically. New vdisks can be assigned to existing AMPs (join procedure) or new AMPs (reconfiguration).

Note: Teradata recommends that you increase memory when you add AMPs.

Adding Vprocs

The following table lists reasons for when to add vprocs.

IF you want to increase …	THEN add …
storage capacity	disks, probably with AMPs, and perhaps nodes. When you add storage, you normally add AMPs. 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. If you add AMPs, be sure your memory is adequate. Normally, you should add memory when you add AMPs. If your existing CPUs cannot handle the load caused by the additional AMPs, you also need to consider adding nodes.

IF you want to increase ...	THEN add ...
single-user response time	AMPs. • 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.
capacity for concurrent sessions	PEs, perhaps nodes, and perhaps a channel connection. • 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. **Note:** 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. • 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 there is not, add nodes.

Adding Memory

When you add memory, you increase the 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, the system is I/O-bound).

The following table lists conditions for when to add more memory.

Condition	Description
Add vprocs to existing nodes	Each vproc consumes 32 MB of memory. When you add vprocs to existing nodes, you probably should add memory. Additional vprocs can substantially reduce free memory, which can cause more I/Os because the system can cache fewer data blocks.
Excessive paging/ swapping (thrashing)	More memory means that more code and data can be cached, achieving less I/O for paging and swapping.
Tables in memory	Increased memory may reduce I/O by accommodating: • Tables that are currently too large to remain in memory during processing • More small tables concurrently residing in memory during processing I/Os can be affected by the size of table data blocks. For information, see the DATABLOCKSIZE option of the CREATE TABLE statement in *SQL Reference: Data Definition Statements*.

Adding Nodes

Often you can satisfy the need for increased capacity or throughput by adding disk arrays, memory, or vprocs. If that does not suffice, you can add one or more nodes.

Although adding a new node costs more than adding disks and memory to an existing node, you must add nodes when your system is CPU bound.

For example, if the configuration is adequate from an I/O perspective but the CPU is at maximum capacity, adding a node tends to alleviate the problem.

If you need more nodes but not storage space, determine if you should add nodes:

- To existing cliques to share existing disk arrays.
- Plus disk arrays to maintain the current ratio.
- Plus AMPs and redistribute data to reduce the amount of storage managed by each AMP.

Reconfiguring Your Teradata Database

When you add nodes, AMPs, or disks to your configuration, you must reconfigure the TPA. The Reconfiguration utility can do most things automatically, such as partitioning the disks and redistributing your data rows.

However, it is good practice to always first archive your data. Also, some types of expansion, such as adding a disk array, still require row redistribution. So if your tables are very large, it may be faster to archive and restore.

You can use the Reconfiguration Estimator utility to obtain an estimate of elapsed time needed for reconfiguration, based on the number and size of the data tables on your current system.

The Reconfiguration Estimator prompts you for information about the planned upgrade and provides estimates for the following phases:

- Redistribution
- Deletion
- Rebuilding secondary indexes

  If you have questions about possible procedures or the reported time estimates, contact the Teradata Support Center. For an overview of the utilities, issues, and procedures involved in a TPA reconfiguration, see "Viewing or Modifying the Teradata Database Configuration" on page 431.

Scaling Your Applications

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 primary index 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 primary index.

The result is many more unique hash combinations, and your applications should continue to perform well as your system expands.

Checking for Cylinder Fragmentation

Fragmented cylinders cause performance degradation. Use the SHOWSPACE command in Ferret to help you determine if after running the PACKDISK command, you need to adjust your Free Space Percentage (FSP).

Note: The Ferret utility is meant for use by Teradata support personnel only. Do not use the utility unless instructed by Teradata Support Center.

From the DBW Supervisor screen, start Ferret and then run SHOWSPACE. The '/l' parameter shows the detail for each logical disk.

To determine what FSP is appropriate for your tables, note how frequently you insert and update rows from your tables. You should set the FSP roughly 5 to 7 percent higher than the percentage of data that will be inserted. Updates will not cause the FSP to increase (cylinder splits) if the rows are fixed length and the DisableWALforDBs (formerly WriteDBsToDisk) setting in the DBS Control record is set to FALSE. If the rows are varchar or compressed, or if DisableWALforDBs is set to TRUE, updates may cause cylinder splits.

To determine if you need to adjust FSP, do the following:

1. Check if you have DisableWALforDBs set to TRUE under DBS Control utility and set it to FALSE if it is. You can check this by entering "display filesys". (DisableWALforDBs defaults to FALSE.)
2. Run the PACKDISK command on the table.
3. Run the SHOWSPACE command on an individual table note the "Avg Utl Per Cyl". The "Avg Utl Per Cyl" is the opposite of the FreeSpacePercent (FSP).
4. Run SHOWSPACE again, after running your insert and update jobs, to check the "Avg Utl Per Cyl". If it has gone down, then the table is getting cylinder splits.

   If you are still getting cylinder splits, this may indicate that your cylinders are fragmented and are splitting because the available contiguous blocks are not large enough to accommodate your blocksize. You may need to consider lowering the blocksize.

   For more information, see Ferret in *Utilities*.

Incident Information Checklist

If you notice your system is having problems and want to contact Teradata Support Center to report an incident, collect as much information about your system as possible. The list below 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 below. It is important to provide any other information that is not listed below but that you feel is important.

1 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 Teradata Database 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 tpareset 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?
- Mini-Cylpacks?
- BNS page pool errors?
- File space problems?
- Memory depletion?
- Lock contention?
- Network congestion, collisions, bottlenecks?
- MVS/VM 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?
- Priority Scheduler configuration?

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 Database system, interfaces, networks, hosts, data marts, intermediate Servers, clients, sessions, users, ODBC, and so on?

7 Describe the Teradata Database System Configuration.

- What are the PTDBMS, PPDE, host, TGTW, PBYNET, PSCSI versions?
- What is the number of nodes, CPU speed, AMPs per node, memory size per node, and storage per AMP?
- Has the Priority Scheduler 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 SCSI, BYNET or other I/O components?

8. What supporting evidence and data do you have to assist Teradata Support Center in troubleshooting?

   - Do you have logs for: System Activity Reporter (SAR), resource usage data, access logging, locking logger, system accounting, canary query, syschk and dbschk, PMON, xperfstate, Teradata Manager, blmstats, tdnstats, streams 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?

SECTION 6 Appendixes

CHAPTER 15 Handling Teradata Crashdumps

The latest information on handling crashdumps is located in the Teradata Database Knowledge Repositories available through Teradata @ Your Service. Sign up for an account at www.teradataatyourservice.com.

Crashdump Articles

Search for “MP-RAS Crashdumps” or “Windows Teradata Crashdump” to access links to crashdump related articles such as:

- How to Upload Teradata on MP-RAS or Linux Crashdumps to the TGSC
- How to Upload UNIX Panic Dumps to the TGSC
- How to load a crashdump that is in DUL format
- Tips and techniques for managing PDE crashdump
- Frequently Asked Questions - Teradata Windows 2000

If you have any questions about crashdumps, contact the Teradata Support Center.

APPENDIX A Teradata Database Configuration, Global Defaults, and Client Connections

This appendix provides an overview of the Teradata Database configuration, the global default controls, and the client connection software, and describes tools you can use to view or change system-wide (global) information.

Viewing the Software Release and Version

From any console or any client session, you can view the currently running Teradata Database version and release level with the following query:

```
SELECT * FROM DBC.DBCInfo;
```

Note: You must have the SELECT privilege on the DBC.DBCInfo 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 1 second.

InfoKey                        InfoData
------------------------------ --------------------------
RELEASE                        12.00.00.00
VERSION                        12.00.00.00
LANGUAGE SUPPORT MODE          Standard
```

On Windows and Linux, you can launch Teradata MultiTool and view the current version of the PDE, DBS, TGTW, TCHN, RSG, and TDGSS.

On MP-RAS, you can run the xctl utility from any console. The utility opens in the Version window, which displays the version number of the currently installed packages.

In addition, the window provides input fields for automatic switching between the PDE and database system software.

For more information on Teradata MultiTool, see *Graphical User Interfaces: Database Window and Teradata MultiTool*. For instructions on using the xctl utility, see *Utilities*.

Reviewing or Changing Global 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 Object (GDO) parameters for:
  - DBS control
  - Gateway control

Object Name	Description	References
DBC.HostsInfo view	Displays the name and assignments of the international character set you defined as the default in the underlying table. You can assign a particular character set as the default for the entire configuration, for a particular client (host), or for a host group.	*International Character Set Support*
DBC.SecurityDefaults	This table contains all the settings for passwords such as when and if passwords expire, the maximum length for the password, and whether or not passwords can use digits or special characters.	• "Customizing Your Password Controls" on page 186. • Security Administration
DBS Control GDO	Establishes throughout the vproc configuration well-known values that you can use for: • Performance tuning • Debugging and diagnostics • Enabling or disabling optional features • Defining global parameters such as cache sizes You can make changes to global settings through the DBS Control utility or the ctl and xctl utilities.	• DBS Control, ctl, or xctl in *Utilities* • *Performance Management*
Gateway Control GDO	Controls the parameters of the current GDO and allows you to control the environment of network connections to a node, including (but not limited to): • Enabling or disabling network logons and external authentication such as Single Sign-On on Windows • Enabling logon from specific IP addresses • Enabling or disabling non-encrypted logon passwords • Defining host groups • Setting reconnect time • Toggling system logs, trace flags, the debugger, and other debugging tools	"Gateway Control Utility" in *Utilities*

Object Name	Description	References
tdlocaledef.txt file	Contains definitions that control the output characters and define the format of locale-dependent data types such as DATE, DECIMAL, TIME, and TIMESTAMP. It also defines the format for currency. **Note:** You can override the global defaults at the user or table level by using the FORMAT phrase in a SELECT query or a CREATE TABLE/VIEW statement. Also note that format changes take effect only after Teradata Database is restarted, and do not affect columns that were created prior to the restart. The location of tdlocaledef.txt depends on your Teradata Database server. See *Utilities* for directory paths.	• "Cultural Format Defaults for Data Types" on page 430 • "tdlocaledef Utility" in *Utilities* • "Output Format Phrases" in *SQL Reference: Data Types and Literals*

Viewing and Changing International Character Set Settings and Defaults

Teradata Database 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 *International Character Set Support*.

Default Server Character Sets

The default server character set is used to store the following information:

- Data Dictionary information comprising object names, such as names of users, databases, tables, columns, macros, views, triggers, functions, and so forth.
- Data Dictionary information other than object names, such as session identifiers, Teradata SQL keywords (for instance, privileges), client platform (host) identifiers, text and syntax captured from queries, and so forth.

The default server character set is also used to store the following information for other users:

- User data (character strings stored in the columns of user-defined data tables), if a specific server character set has not been defined at the user or column level.
- Character valued parameters in macros defined by the user.

Server character sets include:

- LATIN
- UNICODE
- GRAPHIC
- KANJI1
- KANJISJIS

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.

Teradata Database 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.

Teradata Database 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, UTF8, and UTF16 character sets are permanently installed and always available.

Viewing the Status of Client Character Sets

Use the system views described below to view the current status of installed international character sets.

View Name	Description
DBC.CharTranslations	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. **Note:** Because they are permanently loaded and available, the ASCII, EBCDIC, UTF8, and UTF16 character sets are not reported.
DBC.CharSets	Displays the names of the character sets you flagged in the DBC.CharTranslations 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. **Note:** A name in this view and CharTranslations does not prove that it is active. The InstallFlag column in CharTranslations is an indication, but can be misleading if the table was changed without a tpareset.
DBC.HostsInfo	Displays the names of the character sets you assigned as the client defaults, by host.

Client character sets permanently enabled by Teradata Database include:

- ASCII
- EBCDID
- UTF8
- UTF16

For more information on mapping and collation codes, see *International Character Set Support*. For IBM mainframe clients, see "TDP Functionality" on page 442 and *Teradata Director Program Reference*.

For a complete description of the views described in the previous table, see *Data Dictionary*.

Changing Character Set Defaults

You can control 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, the user can find out which client character set is in effect with the SQL HELP SESSION statement[1], and can specify a different (but active) client character in various ways, depending on the particular client software. With the BTEQ client software, you can use the .SET SESSION CHARSET command. For example:

```
.SET SESSION CHARSET ASCII
```

The .SET SESSION command can be entered interactively in the user logon string or at any time during a BTEQ session, or embedded in a BTEQ script file (batch job) or application program.

Default Server Character Set

You can specify the default server character set for a user via the optional DEFAULT CHARACTER SET clause of the CREATE/MODIFY USER statement. If this clause is not specified, the DEFAULT CHARACTER SET for the user is determined by the language support mode.

For example, if the language support mode is Standard, the default server character sets for data submitted by that user is LATIN.

Note: With Japanese Language Mode support, the default character type for user DBC is UNICODE. Previous to Teradata Database 12.0, the default character type was Kanji1.

Changing Collation Defaults

You can define the default collation sequence for a user with the COLLATION clause of the CREATE/MODIFY USER statement. Teradata Database 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 channel-attached clients and ASCII collation for all others.

[1] Use .FOLDLINE ON ALL and .SIDETITLES ON to fit the output for BTEQ on your console screen.

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 Teradata Database collating conventions and instructions on how to define your own sequences, see *International Character Set Support*.

International Language Support Mode

During installation, your database is set up with either Standard or Japanese language support mode enabled. Non-Japanese languages include predefined Chinese, Korean, or Unicode (that is, UTF8 or UTF16) and extended site-defined character sets. Previous to Teradata Database 12.0, the language support mode determined the character set used to store the names of objects in the dictionary tables. However, starting with Teradata Database 12.0 and on, all object names are stored in Unicode so that Data Dictionary field definitions across all Teradata platforms are consistent. So, for example, a field in the Data Dictionary might be stored on disk as VARCHAR (128) UNICODE and processed in memory as CHAR (30) KANJI1/LATIN.

To determine what language support mode your system is supporting, submit the following:

```
sel infodata from dbc.dbcinfo where infokey = 'LANGUAGE SUPPORT MODE';
```

Cultural Format Defaults for Data Types

Tunable defaults for the output format of cultural (locale-specific) data types are globally defined in a Specification for Data Formatting (SDF) text file named tdlocaledef.txt. You can use the tdlocaledef command-line utility to alter this file (See “tdlocaledef utility” in *Utilities*.)

The file provides a method of defining the output format of data types such as DECIMAL, BYTEINT, SMALLINT, INTEGER, REAL, DATE, TIME, and TIMESTAMP. The definition file also allows you to specify numeric group and fraction separator characters, primary and dual currency symbols, and grouping rules and variable-width groups for numbers and text strings.

Note: The global defaults are predefined in the SDF file based on your specifications when you ordered your Teradata Database. The contents of the file are loaded during installation and typically are not changed in a production environment. However, you can redefine them as necessary using the tdlocaledef utility.

However, note that format changes take effect only after Teradata Database is restarted, and do not affect columns that were created prior to the restart.

The global definitions set by the SDF file can be overridden on a per-user basis with the SQL FORMAT phrase. The FORMAT phrase accepts a wide range of directive characters for handling output formats, including:

- Date formats such as order of month, day, and year; or using abbreviated names of days and months in native language

- Twelve-hour time plus order of hour, minute, second, and time zone
- Variable-width groups and separator characters for groups and fractional portions of numbers
- Euro currency format, primary and dual currency names in native format, and currency symbols in native and ISO format

For more details, see “Output Format Phrases” in *SQL Reference: Data Types and Literals*.

Viewing or Modifying the Teradata Database Configuration

A Teradata Database configuration includes the following components:

- Hosts (clients), which can be any combination of multiple:
  - Channel-connected (mainframe) clients
  - Gateway-connected (network) clients
- Parsing Engine (PE) vprocs
- Access Module Process (AMP) vprocs

Each node contains the configurations described in the following table.

This map ...	Describes the ...
current configuration	current arrangement and status of vprocs in the Teradata Database Trusted Parallel Application (TPA).
new configuration	changes and additions to and deletions from the current configuration.

This section explains the configuration maps and how to display or change the configuration, and provides some reconfiguration guidelines.

Configuration Maps

The configuration maps are the working areas in a node. A configuration map:

- Stores the identification and status of each vproc in Teradata Database
- Identifies the AMPs that constitute each AMP cluster
- Identifies each PE and its associated host or host group

The Configuration and Reconfiguration utilities are usually used together to change the contents of the logical configuration. The Configure Teradata operation of the Parallel Upgrade Tool (PUT) utility initiates mapping the logical configuration (for example, AMPs) to the physical configuration (for example, disk storage).

The placement of AMPs and PEs in a configuration is critical to the overall performance of Teradata Database. Consult the Teradata Support Center to resolve questions or issues before starting the Configuration utility, Reconfiguration utility, or PUT utility.

Note: Data rows are associated with AMPs by the hash code of a data row and the hash bucket of an AMP. For more information, see *Database Design*.

Vproc Configurations

You associate PEs defined in the configuration with one or more channel-connected clients or Gateway-connected networks that also are defined in the configuration.

AMPs in a configuration can be related to other AMPs through cluster assignments. Clusters are a technique used with fallback to increase data protection and availability. For details on their assignment, use, and effect on performance, see "AMP Clustering and Fallback" on page 231.

Configuration Activities

When Teradata Database is initialized, the System Initializer utility (sysinit) process builds a default configuration map that describes the target AMP involved in sysinit. This configuration is stored in both the current and new configuration maps.

When the database is operational, the Configuration utility describes the complete system in the new configuration map.

As the system grows and changes, use Configuration to revise the new configuration map to reflect these types of changes to the system:

- Add and delete hosts and AMP and PE vprocs.
- Move data rows from an existing AMP or group of AMPs to a new AMP or group of AMPs (usually on another node). This allows the logical removal of the existing AMP(s) after the data has been successfully moved.
- Move an existing PE or group of PEs to a new PE or group of PEs on another node. This allows for the logical removal of the existing PEs after the PEs have been successfully moved.
- Change host assignments.
- Change cluster assignments.

  Note: It is a good idea to add AMPs before changing clusters. If you do not add AMPs, be sure you have enough permanent space before beginning the reassignment. (CurrentPerm should be less than 53 percent of total MaxPerm; see "Changing Cluster Assignments" on page 234.)

Caution: If Reconfiguration determines there is not enough space to reassign clusters as specified, the process stops. The only way to recover is to re-initialize the system, which will lose all data.

Reconfiguration Activities

Reconfiguration usually involves adding or removing hardware, but sometimes does not (for example, changing cluster assignments). Regardless, after Configuration builds a new configuration map, the Reconfiguration utility redefines the system configuration according to the new map. Reconfiguration copies the new configuration map to the current configuration map and (if AMPs were added) automatically redistributes the data rows.

Note: If Reconfiguration is in the middle of hash map calculations (for row redistribution) and a database reset or system crash occurs, the utility may not be able to restart. A message may appear saying AMP *nn* contains unexpected tables. Clean out the new AMP and resume, as instructed in *Utilities*.

Using the PUT Utility

Before you add a vproc to the current configuration, you first must define it to the physical configuration. Once you delete a vproc from the current configuration, you then must remove it from the physical configuration.

The Move AMPs/PEs operation of the PUT utility initiates the mapping of virtual elements such as the AMPs and PEs to the physical elements, such as disks and host channel or Ethernet controllers. For more information on the PUT utility, see *Parallel Upgrade Tool (PUT) User Guide for UNIX MP-RAS and Linux User Guide* or *Parallel Upgrade Tool (PUT) User Guide for Microsoft Windows User Guide*.

Moving Vprocs for Hardware Upgrades

When you need to replace an outdated node, array, or a down disk, you can use the PUT, Configuration and Reconfiguration utilities. Using these utilities greatly reduces the manual and row redistribution effort required to:

- Remove first-generation nodes and associated storage from a coexistence system when adding a third-generation node.
- Add new storage hardware (disk or disk array)

When replacing old nodes, MOVE of the Configuration utility enables you to:

- Logically move the data rows from an AMP or group of AMPs (with contiguous numbers) on the old node, to an AMP or group of AMPs with higher contiguous numbers on the new node.
- Logically move a PE or group of PEs (with contiguous numbers) on the old node, to a PE or group of PEs with lower contiguous numbers on the new node (as long as any channel-connect restrictions are not violated).

Note: The Reconfiguration utility uses information from Configuration to configure the Teradata Database components into an operational system.

For more information, see “Replacing Old Nodes and Storage” in the PUT utility user guide.

Typical Reconfiguration Procedures

The following table lists some typical procedures for reconfiguration.

IF you want to ...	THEN ...	For instructions ...
add a hard disk	completely reconfigure your system by doing the following: **1** Create a tape archive of the data. **2** Re-partition the disk. **3** Update the Teradata Database configuration, adding AMPs if necessary. **4** Run RECONFIG to redistribute data across all AMPs. Or if you need to more quickly configure the system and the data is not as crucial, do the following: **1** Create a tape archive of the data. **2** Re-partition the disk. **3** Update the Teradata Database configuration, adding AMPs if necessary. **4** Run SYSINIT. **5** Restore the data.	see the following in *Utilities*: • Configuration Utility to add AMPs • Reconfiguration Utility to run RECONFIG • System Initializer Utility to run SYSINIT Use PUT to change the configuration of your system. See: • *Parallel Upgrade Tool (PUT) User Guide for UNIX MP-RAS and Linux User Guide* • *Parallel Upgrade Tool (PUT) User Guide for Microsoft Windows User Guide*
view the current and any newly defined logical and physical configuration	use the: • LIST command of the Configuration utility, on any Teradata Database • HARDWARE command of the: • ctl utility on Windows and Linux • xctl utility on UNIX MP-RAS (to display the PDE hardware configuration) • Teradata Manager configuration displays • DBS console window configuration displays	see the following: • *Utilities* • *Teradata Manager User Guide* • *Graphical User Interfaces: Database Window and Teradata MultiTool*
obtain an estimate of the elapsed time needed for reconfiguration	use the Reconfiguration Estimator utility. Basing calculations upon the number and size of the data tables on your current system and your response to prompts about planned changes, the Estimator calculates the time needed for: • Reconfig redistribution phase • Reconfig deletion phase	contact the Teradata Support Center if you have questions on estimated time or other parts of this procedure. See the following chapters in *Utilities*: • “Reconfiguration Estimator” • “Vproc Manager”

<table>
<tr><th>IF you want to ...</th><th>THEN ...</th><th>For instructions ...</th></tr>
<tr><td>change the TPA configuration in any way</td><td>place the system into a stopped state before configuring the system:
1 Disable logons (see "Changing Logon States and Restarting the System" on page 291).
2 Run Vproc Manager utility to check the status of each AMP and PE
3 Prepare the PDE and hardware by making sure the PDE and all nodes are in the NULL state.
4 To verify the PDE state from Database Windows, enter:
<code>pdestate</code>
The correct stopped status display is:
• On Windows:
<code>PDE state is DOWN/HARDSTOP</code>
• On UNIX:
<code>Parallel Database Extension state is NULL/ STOPPED</code>
• On Linux:
<code>PDE state is STOP/KILLTASKS.</code>
<b>Note:</b> For UNIX systems, if the node is a standby node, the pdestate command reports the node as NULL/STANDBY and for Windows systems, a standby node may appear as DOWN/HARDSTOP.</td><td>contact the Teradata Support Center if you have questions on estimated time or other parts of this procedure.
See the following chapters in <i>Utilities</i>:
• "Reconfiguration Estimator"
• "Vproc Manager"</td></tr>
<tr><td>add, replace, or delete a node or disk storage units</td><td>stop the system and all AMPs, PEs, logons, PDE, and nodes. When everything is quiescent, perform the following:
1 Physically add the new node or storage to the hardware platform.
2 Run PUT, using the CONFIGURE TERADATA operation to create vprocs on the new node or vdisks on the new storage.
3 Run the Configuration utility. The operation depends on the function being performed.
<b>Note:</b> It is possible to combine the ADD and MOVE operations and perform both functions at the same time.
<table>
<tr><th>IF you are ...</th><th>THEN use the ...</th></tr>
<tr><td>adding (not replacing) a new node or disk</td><td>ADD AMP and ADD PE commands to add the item created in Step 2 with PUT.</td></tr>
<tr><td>replacing a node or disk</td><td>MOVE or ADD/MOVE PE/AMP commands to swap identifiers between the old and new vprocs.</td></tr>
</table></td><td>first, contact the Teradata Support Center; then see:
• <i>Utilities</i> for:
  • "Configuration Utility"
  • "CheckTable Utility"
  • "Reconfiguration Utility"
• <i>Parallel Upgrade Tool (PUT) User Guide for UNIX MP-RAS and Linux User Guide</i>
• <i>Parallel Upgrade Tool (PUT) User Guide for Microsoft Windows User Guide</i>
• For node and storage removal, your server hardware maintenance manual</td></tr>
</table>

IF you want to ...	THEN ...	For instructions ...

add, replace, or delete a node or disk storage units (continued)	**4** Immediately before performing a reconfiguration, run the CheckTable utility at LEVEL PENDINGOP to make sure that no tables are in the `pending xxx` state by doing the following:
	a If the system is active, run the CheckTable utility for all tables using concurrent mode. For example:
	`AT LEVEL PENDINGOP PRIORITY = H CONCURRENT MODE;`
	Otherwise just use the following options:
	`AT LEVEL PENDINGOP SKIPLOCKS PRIORITY=H;`
	Note: Make sure logons are disabled. A hang will occur if you run SKIPLOCKS IN PARALLEL with PENDINGOP while users are logged on.
	For concurrent mode, if you need to check the Data Dictionary as well, execute a separate CHECK DBC AT LEVEL PENDINGOP CONCURRENT MODE command.
	b Visually audit the CheckTable output to identify any skipped tables. (If you select the ERROR ONLY option, CheckTable will display only the tables with warnings and errors and skipped tables.)
	c Run CheckTable again for each table that was skipped because of lock contention. Resolve the lock contention.
	5 Run the Reconfiguration utility:
	• For an add-only operation: Redistribute the data rows across the new and existing vdisks.
	• For a replace operation: Physically copy the contents of the old pdisks to the new pdisks.
	• Copy the new configuration map to the current configuration map.
	Note: When Reconfig starts, an added AMP shows a status of NewReady, and remains in this state until PUT removes it from the configuration.
	6 Use PUT CONFIGURE TERADATA to remove the old node or old vproc definitions from the configuration.
	Note: When processing is complete, the old AMPs should not appear in a configuration or status list.
	7 Check the state of your databases and data by doing the following:
	a Run the full SCANDISK command (from the Filer utility) with all either logons enabled or DBC logons enabled.
	b Run CHECKTABLE LEVEL 3 on database DBC with logons disabled (quiescent system).
	c Run CHECKTABLE LEVEL 2 on all critical production databases and tables. If all logons are disabled, it is safe to use the IN PARALLEL option to reduce CheckTable runtime.
	Note: CheckTable with IN PARALLEL requires substantial spool space.
		first, contact the Teradata Support Center; then see:
		1 *Utilities* for:
		• “Configuration Utility”
		• “CheckTable Utility”
		• “Reconfiguration Utility”
		2 *Parallel Upgrade Tool (PUT) User Guide for UNIX MP-RAS and Linux User Guide*
		3 *Parallel Upgrade Tool (PUT) User Guide for Microsoft Windows User Guide*
		4 For node and storage removal, your server hardware maintenance manual.

IF you want to ...	THEN ...	For instructions ...
add, replace, or delete a node or disk storage units (continued)	**8** Visually audit the CHECKTABLE output to identify any skipped tables. If you select the ERROR ONLY option, CheckTable will display only the tables with warnings and errors. Examine the CheckTable output to identify skipped or problematic tables. **9** Run CheckTable for each table that was skipped because of lock contention. **10** When all tables are checked, physically remove the old node or storage from the hardware platform.	

Client Configuration Overview

There are two types of Teradata Database clients.

Type	Description
Channel attached	Mainframe computing system, such as IBM VM.
Network attached	Windows or UNIX workstation.

A session is the transfer of data between a user on a client and Teradata Database on the server, including the logical and physical connections that make data transfer possible.

Teradata Client Management Subsystems

During a session, all communication between a user and the database pass through a client-resident management subsystem called:

- Teradata Director Program (TDP) on a channel-attached client
- Micro Teradata Director Program (MTDP) on a network-attached client

Director program functions, along with security, include:

- Session initiation and termination
- Logging, verification, recovery and restart notification for client applications
- Coordinates the physical input to, and output from, Teradata Database

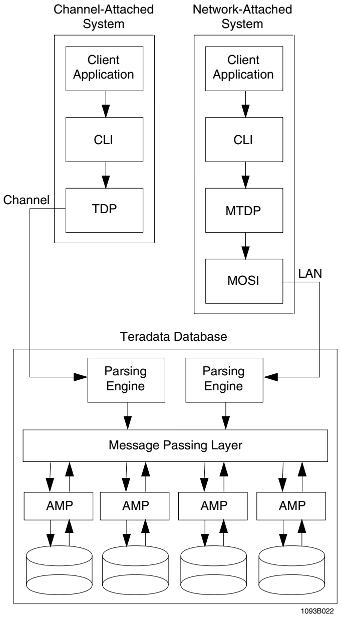
Teradata Director Program (TDP) for Channel-Attached Clients

The channel interface enables communication between a mainframe client and Teradata Database. TDP establishes a session with the associated PE and manages communications between the client and Teradata Database. When a user or application submits an SQL request, the TDP transfers it to the Teradata Database, receives the response, and transfers the response to the application.

Micro Teradata Director Program (MTDP) for Network-Attached Clients

In a network environment, MTDP 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 channel-attached and network-attached clients, and Teradata Database.



Communicating with Teradata Database

What is a Session?

A session is a logical connection between the user and Teradata 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 Teradata Database and is established when the Teradata 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 Parcels

Once a session has been logged on, Teradata Database communicates with the host through request and response parcels. (Response parcels are sometimes called data parcels because they contain the data requested by the client). The parcels have a header and a body depending on which parcel stream is read and interpreted by the receiving components. The maximum response parcel size limit is 1MB.

How Request Parcels Are Handled

To access the Teradata Database from a mainframe client, the user or application program logs on through a Teradata interface facility (such as BTEQ) and submits a request. The request is processed by the interface and directed to a Teradata Director Program (TDP).

Each request is handled as follows:

- In a channel environment, the TDP builds the request parcel and sends it through the channel to the PE associated with that TDP.
- In a network environment, the MTDP builds the request parcel and sends it over the network to the gateway. The gateway distributes the parcel to an available PE on the node on which the gateway is running.

Response Parcels

The result of a query or a status becomes the returned answer set. The PE turns the answer set into a response parcel and returns it to:

- The TDP, in a channel environment. The TDP returns it to the client utility or program.
- The gateway, in a network environment. The gateway sends it to the MTDP, and the MTDP returns it 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 preprocessed by the Teradata Preprocessor2, which runs on a client system.

You can run the preprocessor against an application without connecting to the Teradata Database if you specify the SQLCHECK (-sc) option as NOSYNTAX. However, to precompile and run an application, a separate connection to Teradata Database is required for the precompilation and the runtime event.

Also offered is a client utility, the Data Definition Language Processor (DDLP), that processes ANSI data definition language statements for Teradata Database for UNIX. This processor breaks down entry level ANSI SQL CREATE SCHEMA statements into their individual components and sends requests to the Teradata Database to create databases, tables, views, and user privileges. DDLP commands can be entered from a workstation, terminal, or command file.

For instructions on how to prepare, precompile, and run an application on Teradata Database with embedded SQL statements, see *SQL Reference: Stored Procedures and Embedded SQL* and *Teradata Preprocessor2 for Embedded SQL Programmer Guide*.

For details on using DDLP and CLIv2, see:

- *Teradata Data Definition Language Processor Reference*
- *Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems*
- *Teradata Call-Level Interface Version 2 Reference for Channel-Attached Systems*

Channel Environment

Teradata utilities and software programs support Teradata 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 Channel Interface enables communication between a mainframe client and a Teradata Database server using a channel with either a parallel or serial I/O interface. Two devices that consist of an even and odd address pair make up this connection. Each device is independent of the other. The client sends Teradata Database messages to a server using the device with the even address. The server returns responses to the client using the device with the odd address.

This hardware was originally known as the Channel Interface Controller (CIC). Each CIC was associated with one Teradata Interface Processor (IFP) to provide a device pair on a parallel I/O interface channel, or with the MicroChannel-to-Channel Adapter (MCCA) board Application Processor (AP) and a Parsing Engine (PE).

The current platforms can be connected with either traditional Bus and Tag or the new ESCON fiber channel. The Host Channel Adapters available on Teradata servers are:

- PBSA—PCI Bus ESCON Adapter (ESCON fiber)
- EBCA—EISA Bus Channel Adapter (Bus and Tag)
- PBCA—PCI Bus Channel Adapter (Bus and Tag)

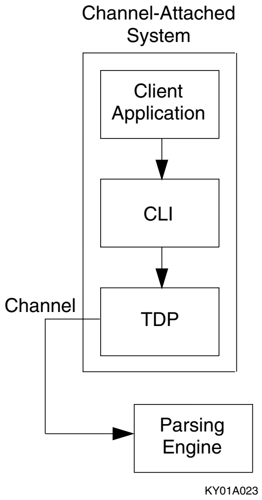
Depending on your workload and required throughput, you do not need to configure one HCA per node. In smaller systems (up to 4 or 6 nodes), you might configure 2 or 3 HCAs. In a two-node system, however, you would configure one HCA per node for redundancy.

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 channel-attached clients that play important roles in getting requests to and from the Teradata Database.



The following table describes these software components.

Component	Description
Client application	• Written by a developer in your company • Written under contract to you by Teradata • One of the Teradata Database-provided utilities Users use these applications to submit SQL statements, maintain files, and generate reports. For details on supported programming languages, the Teradata Preprocessor2, and embedded SQL requirements, see *Teradata Preprocessor2 for Embedded SQL Programmer Guide.*
Call-Level Interface Version 2 (CLIv2)	A low-level interface to the Teradata Database. It consists of system calls that: • Create sessions • Allocate request and response buffers • Create and deblock parcels of information • Fetch response information for the requesting client. For more information, see *Teradata Call-Level Interface Version 2 Reference for Channel-Attached Systems* or *Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems.*
Teradata Director Program (TDP)	Manages communication between mainframe clients and Teradata Database. Functions include: • Session initiation and termination • Logging, verification, recovery, and restart of client applications • 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) See "TDP Functionality" on page 442 and *Teradata Director Program Reference*.

Channel 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 MVS or VM client or multiple clients. A session number and a logical client number identify each session to the MVS or VM 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 Teradata Database server as a unit, using the same logon string via a TDP START POOL command.

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.

TDP Functionality

All messages that a mainframe client sends or receives to Teradata 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 percent of the cells of a certain size are in use.

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 MVS console
- MVS/TSO users
- The VM console
- VM/CMS virtual machines
- CLIv2 applications

Messages that result from executing operator commands entered from a console are returned to the console.

For detailed information, see *Teradata Director Program Reference*.

Network Environment

In a network environment, each workstation on the network has a copy of Teradata client software, including the utilities, programs, and drivers needed to access Teradata Database.

Functionality

The elements that make up the Teradata client software in a LAN-attached environment are:

- MTDP—Micro Teradata Director Program
- CLI—Call Level Interface
- MOSI—Micro Operating System Interface

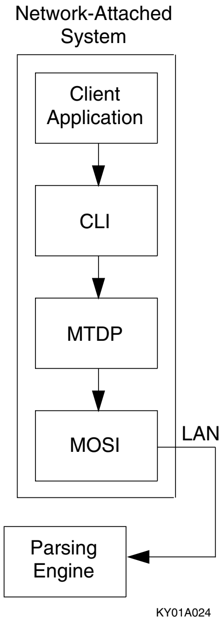
In a network environment, MTDP and gateway software in the node handle the functions that TDP handles in a channel-attached environment.

A network interface card connects workstations directly to the network, and an Ethernet card in the MCA bay connects the network directly to a node. These connections provide workstation access to the gateway software in the node. Redundancy is provided with two separate connections to the network.

There are two network cards in the MCA bay and two Ethernet cables to the network.

Software Components

The figure on the following page illustrates the software components in network-attached clients that play important roles in getting and executing requests to and from Teradata Database.



The following table describes these software components.

Component	Description	References
Client application	Either: • Written by a developer in your company • Written under contract to you by Teradata • One of the Teradata Database-provided utilities Users use these applications to submit SQL statements, maintain files, and generate reports.	*Teradata Preprocessor2 for Embedded SQL Programmer Guide*
Teradata call-level interface (CLI)	A low-level interface to the Teradata Database, CLI consists of routines that perform functions similar to CLI system calls on channel-attached clients, including: • Logging sessions on and off • Submitting SQL queries • Receiving responses with the answer set	• *Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems* • *Teradata Call-Level Interface Version 2 Developers Kit for Microsoft Windows*
Micro Teradata Director Program (MTDP)	Library of session management routines that perform for network-attached clients many of the TDP functions, such as session control and parcel transmission.	• *Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems* • *Teradata Preprocessor2 for Embedded SQL Programmer Guide* • *Teradata Parallel Transporter Operator Programmer Guide*
Micro Operating System Interface (MOSI)	Library of routines that provide operating system independence for client access to the Teradata Database.	• *Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems* • *Teradata Parallel Transporter Operator Programmer Guide* • *Teradata Driver for the JDBC Interface User Guide*
Database connectivity drivers	A variety of drivers for open connectivity products, such as ODBC and JDBC.	*Teradata Driver for the JDBC Interface User Guide*

The Network Gateway

The gateway software is the interface between the network and the Teradata Database. It runs on the Teradata Database server as a separate operating system task. Client sessions that communicate through the gateway to the Teradata Database may be resident on the Teradata server, or may be installed and running on network-attached workstations.

In contrast, sessions originating from a channel-attached mainframe access the Teradata Database through channel connections and TDP software, and bypass the gateway completely.

The following table describes tools available to administer your gateway.

Tool	Name	Function
Gateway Control Globally Distributed Object	GTWCONTROLGDO	Defines the current values of gateway defaults throughout all nodes in the Teradata Database configuration.
Gateway Control Utility	gtwcontrol	Recommended for use only by your Teradata field support engineer, to: • Control the defaults in the Gateway Control GDO, such as the number of sessions per gateway. (Typically, these are set only by an Teradata field support engineer. Contact TSC if you wish to change this setting.) You can set options such as connection time out length, external authentication settings, or whether to use user settable defaults or standard defaults when the system is reconfigured, and host groups or gateway vprocs are added, and more.
Gateway Control Utility (continued)	gtwcontrol	Recommended for use only by your Teradata field support engineer, to: • On Windows and Linux, determine how you define the names of Teradata users and how they log on. For details, see: • "Single Sign-On" on page 188 • "Diagnosing Logon Encryption Errors" on page 398
Gateway Global Utility	gtwglobal	• Monitor network sessions and traffic • Disable logons • Force off a rogue session • Investigate gateway problems

Using Host Groups to Manage Gateways

On UNIX MP-RAS, the gateway runs on the control vproc of the node and not on its own vproc. As a result, there can only be one gateway per node. On Windows and Linux, however, the gateway runs in its own vproc and therefore there can be multiple gateways on a node.

All the gateways that belong to the same host group manage a set of PEs. The set of PEs used to manage sessions is determined by the host number in the mapping of the host group as defined by the Configuration utility. For more information on host group assignments, see Gateway Control Utility in *Utilities*.

The advantage of defining multiple host groups is that if you submit very similar SQL requests and have a separate host group to process those requests, you could get a better cache hit rate. By controlling where the PEs and gateways for a host group are located and controlling which jobs go to which host groups, it may better balance the gateway and PE workload.

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.

Controlling Network Sessions

Use the Gateway Global utility to find the information described in the following table.

Command	Description
DISABLE LOGONS	Disable logons to Teradata Database through the gateway. This includes logons through DBC.
ENABLE LOGONS	Enable logons to Teradata Database via the gateway.
DISCONNECT USER	Disconnects all sessions owned by a user.
DISCONNECT SESSION	Disconnects a specific session. Must provide the session number in the command syntax.
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.

For More Information

For information on the Gateway Control or Gateway Global utilities, see *Utilities*. For information on how to configure your Gateway, see *Parallel Upgrade Tool (PUT) for UNIX MP-RAS and Linux User Guide* and *Parallel Upgrade Tool (PUT) for Microsoft Windows User Guide*.

APPENDIX B Import/Export Utilities

This appendix provides information to help you decide which import/export utility to use. It also discusses how to monitor import/export jobs. However, because system configurations differ, the suggestions in this appendix are generalizations. It is always best to run tests on an adequate amount (approximately 10 percent) of real data before settling on a utility choice.

Client-based Utilities

The Teradata Database includes two different types of utilities: client-based utilities and AMP-based utilities. Client-based utilities are installed on a client system. The term *client* may refer to a channel-attached client or a network-attached client. Client-based utilities run under the client operating system.

You initialize AMP-based utilities using the Teradata Database Window. On a channel-attached mainframe, a console interface called Host Utility Console (HUTCNS) provides access to a number of AMP-based utilities. You may also use Teradata Manager Remote Console or cnsterm.

The Import/Export Utilities discussed in this appendix are client-based utilities. These are:

- Basic Teradata Query (BTEQ)
- FastLoad (FLoad)
- FastExport
- MultiLoad (MLoad)
- Teradata Archive/Recovery Utility (ARC)
- Teradata Parallel Data Pump (TPump)
- Teradata Parallel Transporter

For the most up-to-date information on capabilities, usage rules, limits, and restrictions, refer to the respective client-based utility manuals.

Basic Teradata Query (BTEQ)

BTEQ exports and imports data between a client and the Teradata Database. In addition, BTEQ provides report writing capabilities. (For detailed information, see *Basic Teradata Query Reference*.)

Use the capabilities of BTEQ described in the following table to improve performance.

IF you are performing...	THEN...
full table scans (all-AMP operation)	use a single session.
operations that require less than all-AMPs	use multiple sessions.
Primary Index (PI) operation	determine empirically the optimum number of sessions. More sessions do not necessarily improve performance.
operations that do not require statement accounting to be returned to your output file	use .SET QUIET ON in BTEQ.
large exports over a network	update the `resp_buf_len` entry in the */usr/lib/clispb.dat* file from 8 Kbytes to 31 Kbytes.

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 channel-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 network-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.

For more information on BTEQ, see *Basic Teradata Query Reference*.

FastLoad

FastLoad (FLoad) allows you to load data from the client to your Teradata Database system. It allows fast loading of a single empty table that has *no* SIs, JIs, HIs, or referential integrity (FKs).

FLoad loads large amounts of data into an empty table on the Teradata Database. However, row size and the number of columns, more than any other factors, do affect performance.

Because it is not usually an option to reduce the size of rows and columns, take the things listed in the following table into consideration when you load large tables.

IF you are loading a large table...	THEN...
with fallback	• 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.
without fallback	• 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 15. The default is 5. (See *Utilities* for more information.)

For more information on this utility, see *Teradata FastLoad Reference.*

FastExport

FastExport quickly exports data from a Teradata Database 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 15[1]. The default is 5.

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.*

MultiLoad

MultiLoad (MLoad) allows you to upload data from the client to your Teradata Database. It operates on multiple tables simultaneously and can also insert, update, and delete data.

The maximum number of MLoad jobs you can run is 15[2]. The default is 5. (See *Utilities* for more information.)

[1] The Teradata Support Center can help modify your system to run more than 15 concurrent FastExport jobs if they have determined that this does not negatively impact the performance of your system.

[2] The Teradata Support Center can help modify your system to run up to 30 concurrent MultiLoad jobs if they have determined that this does not negatively affect the performance of your system.

You can also change the maximum number of load jobs allowed by defining throttle rules through Teradata DWM.

To improve performance:

- Target data tables without triggers or join indexes.
- 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.

MLoad does not support the following:

- Aggregate operators
- Exponential operators
- Arithmetic functions
- Concatenation of data files
- Hexadecimal forms
- FKs
- USIs
- JIs
- HIs

For more information, see *Teradata MultiLoad Reference.*

You can alternatively use MERGE instead of MultiLoad for bulk data loading to take advantage of performance benefits, avoiding a restart, and keeping USIs and triggers. For more information, see the Teradata Database Orange Book titled “Exploring the Benefits of Teradata 12.0 – ANSI MERGE Enhancements".

Teradata Parallel Data Pump (TPump)

The TPump utility allows real time updates from transactional systems into the 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, on the Teradata Database.

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*.

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 Teradata Database guarantees either completion or rollback of all statements packed in a 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*.

Teradata Archive/Recovery Utility (ARC)

The ARC utility is used to:

- Archive selected partitions, tables, databases, and certain dictionary tables
- Reload data onto the same or a different Teradata Database
- Checkpoint, rollforward and rollback permanent journals (data recovery)

The ARC utility, through its associated script language, provides the link between Open Teradata Backup (OTB) solutions and the Teradata Database.

In general, the direction with all OTB solutions is to provide automation in the generation of ARC scripts, requiring less detailed knowledge of the ARC utility to perform the typical archive, restore, and recover tasks. Each OTB provides varying degrees of support towards this direction. The Teradata Database Administrator will need a detailed understanding of ARC when it is necessary to analyze problems in archive, restore, and recover operations or it is necessary to develop ARC scripts to perform operations that are not supported by the particular OTB's automated script generation.

Although you can perform ARC functions with other utilities, you can use ARC to unload and reload databases and tables on the same Teradata Database, or to move your databases from one Teradata Database version or platform to another. For more detail, see Chapter 9: “Archiving, Restoring, and Recovering Data.”

To improve ARC performance, you can:

- Run multiple cluster jobs in parallel for large tables and databases.
- Run multiple selected partition jobs for large PPI tables.
- Use RESTORE rather than COPY. Use COPY only when RESTORE does not perform the required function, such as restoring to a different Teradata Database configuration. (Compare the RESTORE and COPY command in *Teradata Archive/Recovery Utility Reference*.)
- Use the appropriate number of sessions for the job:

Job	Comment
All-AMPs archive	Specify no more than one session/AMP.
Cluster or specific archive	Specified sessions should be an even multiple of the total AMPs, for example, 4 AMPs, use 4, 8, 12, and so forth.
Restore/copy	This job uses all of the sessions specified in the SESSIONS parameter.

Note: When working with RI constraints or PPIs, RESTORE invalidates table headers. For an explanation of how to find the error rows and validate the state of the table, see "Solving PPI and RI Validation Errors" on page 411.

For more information on the ARC utility, see *Teradata Archive/Recovery Utility Reference*.

Teradata Parallel Transporter

Teradata Parallel Transporter (Teradata PT) is an object-oriented software system that executes multiple instances of data extraction, transformation, and load functions in a scalable, high speed parallel processing environment. It combines and expands on the functionality of the other utilities (Fload, MLoad, FastExport, and TPump) into a single product through the use of a single scripting language.

You can invoke Teradata PT through scripts or with the Teradata PT API. Third-party applications can execute Teradata PT operators so that data extraction and load capabilities can be extended.

You can obtain data from multiple dissimilar data sources for loading into Teradata as well as extract data from the Teradata Database to flat files, access modules, and ODBC-compliant sources.

For more information, see *Teradata Parallel Transporter User Guide*. The guide lists other TPT publications such as references, programmer guides, and user guides.

Interpreting Teradata Manager LOAD Utility Status

When you use Teradata Manager to observe the progress of load, archive, export, and other jobs, remember the following:

- The system uses Logon Sequence Number (LSN) to calculate the impact of all sessions.
- All work the system performs during a session is charged to that session. Teradata Manager displays system resource usage on a session basis.

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.

Loading Data into Teradata Database

You can load your production database using one of the tools described in the following table.

Method	Comments
FastLoad or MultiLoad utility	These utilities populate empty tables using data from a client file or sequential data set. You can operate on 5 to 30 tables simultaneously (depending on DBSControl settings) and can manipulate data in existing tables. However, MultiLoad cannot load tables with USIs, and FastLoad cannot load tables with any SIs.
TPump utility	TPump uses standard SQL DML operations, not block transfers. TPump maintains up to 60 tables at a time. You can specify the percentage of system resources to use for an operation. This allows background maintenance to best suit the processing resources any time in the day.
INSERT requests	Submit the INSERT requests using one of the following methods: • An application program • A macro • One or more BTEQ sessions (with or without repetition) BTEQ can insert either specified values or, with variables you define in a USING request modifier, data from a disk file or sequential data set.

Considerations When Loading PPI Tables

Client utilities support PPI tables with these restrictions and recommendations).

IF you plan to use ...	THEN be aware that the utility ...
MultiLoad IMPORT	• Does not operate on tables with USIs (and NUPPI tables may be defined with USIs) • Does not support updates of PI columns or partitioning columns • Requires values in all PI fields and partitioning fields
FastLoad	does not support SIs of any kind (and many NUPPI tables are defined with USIs).
TPump	can handle everything but performs better if you: • Provide values for all the primary index and partitioning columns (to avoid lock contentions) • Do not update the PI or the partitioning columns

Choosing the Best Utility

Some of the client utilities have similar functions. However, in order 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 percent of total rows
Moderate number of rows	20 - 30 percent of total rows
Large number of rows	40 - 50 percent 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.

Task	First Choice	Second Choice
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.	

Task	First Choice	Second Choice
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. MLoad 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 will beat an SQL delete in many cases as there is no rollback logging. However, MLoad must complete or else the table will remain in an unstable state. For MLoad to work best, drop secondary indexes, run the MLoad 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
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

Task	First Choice	Second Choice
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 Teradata Database	ARC	FastExport and FastLoad
Copy all tables in the Teradata Database to a different Teradata Database system	ARC	
Load an identity column table	TPump (in ROBUST mode)	BTEQ .IMPORT

Monitoring a FastLoad or MultiLoad Job

This section describes how the nature of work performed during the data loading phase and the table processing phase of a FastLoad or MultiLoad job determines where and when the work is charged to the different sessions. Consequently, the pattern of work charged to different sessions causes Teradata Manager to display resource usage of data in a particular way.

Logon Sequence Number

When you run a FastLoad or MultiLoad job, you are logged onto a Teradata SQL session and also under the same user name to *n* number of FLoad or MLoad sessions. The LSN is associated with each session when it logs on and identifies a collection of sessions performing the same job. All sessions that are associated with the same job—whether Teradata SQL, FLoad, or MLoad sessions—are logged on as the same user and have the same LSN. If you want to see the total impact of a FastLoad or MultiLoad job, you must calculate the total impact of all sessions reported with the same LSN.

Resource Usage

For those sessions in which work is charged to AMPs, Teradata Manager displays the following resource usage:

- AMPState or PEState is non-idle.

  This typically means that AMPState may be ACTIVE or BLOCKED.

  For Teradata SQL sessions, PEState may be PARSING-WAIT, PARSING, ACTIVE, or BLOCKED. For FLoad or MLoad sessions, PEState is always UNKNOWN.
- AMPCPUSec and AMPIO (logical I/O) show resource usage.

Understanding that these values are affected by the charging scheme helps explain the impact of resource usage or work charged in the next two tables.

Resource Usage in a FastLoad Job

The following table describes the two primary phases of a FastLoad job. In Phase 1, the system completely loads data from the client to the individual AMP associated with each FastLoad session. The system performs most of the work during the *n* FastLoad sessions and charges that work to the AMPs.

The system also performs a small amount of setup work for the operation during the Teradata SQL session. When the system is not performing this setup work, AMPState or PEState looks idle.

Phase 2 is the end loading phase, when the system completes data transfer to the AMP from the client, and the AMPs insert data into the table. The system performs work mainly during the Teradata SQL session and charged primarily to the AMPs. Although AMPs or PEs are accumulating data in the Teradata SQL session, you may not notice the accumulation of data for a few sampling periods, depending on the size of the FastLoad job or the sampling rate.

FastLoad Phases	Start of FastLoad Phase	1 Teradata SQL Session	*n* FASTLOAD Sessions
Phase 1 (data transfer)	When system starts to process BEGIN LOADING command	• The system performs a small amount of work. • AMPState or PEState looks IDLE most of the time and shows resource usage.	• System charges work to AMPs. • In Teradata Manager, AMPState or PEState is non-idle most of time; AMPCPU and AMPIO show resource usage.
Phase 2 (inserting, error checking)	When system starts to process END LOADING command	• System charges work to AMPs. • In Teradata Manager, AMPCPU or PECPU and AMPIO show resource usage.	

Phase 1 and Phase 2 can be executed in separate jobs. It is possible to execute multiple Phase 1 jobs, feeding data into one table, followed by one Phase 2 job.

Resource Usage in a MultiLoad Job

The following table describes the two primary phases of a MultiLoad job.

- In the acquisition phase, the system loads data from the client to the individual AMP associated with each MLoad session. The system performs most of the work during the MLoad sessions and charges it to the AMPs. The system also performs a small amount of setup work for the operation during the Teradata SQL session. When the system is not performing this setup work, AMPState or PEState looks idle.
- In the application phase, the system completes data transfer to the AMP from the client, and the AMPs process the various insert, update, and delete operations on the designated rows in the specified table. The system performs work mainly during the Teradata SQL session and also charges the work to the AMPs. Although AMPs or PEs are accumulating data in the Teradata SQL session, you may not notice the accumulation of data for a few sampling periods, depending on the size of the MultiLoad job or the sampling rate.

MultiLoad Phases	Start of MultiLoad Phase	1 Teradata SQL Session	*n* MLoad Sessions
Acquisition Phase	MultiLoad Acquisition Phase	• The system performs a small amount of work. • AMPState or PEState looks IDLE most of the time and does not show resource usage.	• System charges work to AMPs. • In Teradata Manager, AMPState or PEState is non-idle most of time; AMPCPU and AMPIO show resource usage.
Application Phase	MultiLoad Application Phase	• System charges work to AMPs. • In Teradata Manager, AMPState or PEState is non-idle most of time; AMPCPU or PECPU and AMPIO show resource usage.	

Data Transfer Phase

The following table describes work is performed in two different partitions during the data transfer phase of a FastLoad or MultiLoad job.

Partition	Description
Teradata SQL	The system logs on one session under this partition. This session: • Parses FastLoad or MultiLoad commands • Performs security checking • Controls the process of executing the FastLoad/MultiLoad job During the data transfer phase, the session processes CHECKPOINT commands that verify that previously transmitted data has been written to disk on the Teradata Database. Note that for the Teradata SQL session, even though the client (or host) does not have an outstanding request, AMPs show usage of Central Processing Unit (CPU) and Input/Output I/O resources.

Partition	Description
FastLoad or MultiLoad	Under this partition, the system logs on *n* sessions under this partition, where *n* is greater than zero and less than or equal to the number of AMPs. These sessions each have a RunProcId corresponding to an AMP, and perform the initial data loading process. Once the system transfers all data to the AMPs, these sessions become idle. Note that when the data transfer phase complete, the AMPCPUSec and AMPIO values remain constant at the last value reported before data transfer completes on the FLoad or MLoad sessions. Your FastLoad or MultiLoad job is not hung because no increment in values has occurred.

Work is charged to different sessions during the data transfer phase. The following types of work are performed:

- On AMPs that initially receive blocks of data transferred from the host (or client):
  - Translate the data into internal Teradata Database format.
  - Perform some legality checking.
  - Transmit the data to the AMP where it will be permanently stored.

  The system charges resources to the individual FLoad or MLoad partition sessions.
- On the AMPs where the data will be stored, that is, the AMPs which own the rows based on the Hash of the PI fields of the table:
  - Collect the data rows into a buffer.
  - Write the data to disk when the buffer is full.
  - Process the CHECKPOINT operations periodically generated by the FastLoad or MultiLoad application. The system charges resources to the Teradata SQL partition session.

As far as the host (or client) is concerned, during the time data is transferred to the Teradata Database, the Teradata SQL session does not have any outstanding requests. Thus, for the Teradata SQL session, the PEState is IDLE and the AMPState is ACTIVE, showing the consumption of large amounts of CPU and I/O resources. The only time a request is outstanding on this session is during the brief periods when the system is running a CHECKPOINT operation.

Monitoring an ARC Job

This section describes how the nature of work done during the data loading phase and the table processing phase of an ARC job determines where and when the work is charged to the different sessions.

Consequently, the pattern of work charged to different sessions causes Teradata Manager to display resource usage of data in a particular way.

Logon Sequence Number (LSN)

Like FastLoad and MultiLoad, the system logs on all sessions associated with the same ARC job as the same user name and with the same LSN.

LSN is associated with a session when the system logs on, and identifies a collection of sessions performing the same job.

To determine the total impact of an ARC job, calculate the total impact of all sessions reported with the same LSN.

Resource Usage Charges

The system charges work or the part of the job done in a session partition to that partition. For those sessions in which work is charged to AMPs, the following resource usage is displayed in Teradata Manager:

- AMPState is non-idle.

  This typically means that AMPState may be ACTIVE or BLOCKED.
- AMPCPUSec and AMPIO (logical I/O) show resource usage.

The following table shows the primary resource usage and where the work is charged for the three major operations carried out in an ARC job.

ARC Commands	One SQL Session	*n* HUTPARSE Sessions	One HUTCTL Session
DUMP (ARCHIVE)	• The system performs a small amount of work. • AMPState or PEState looks IDLE most of the time and does not show resource usage.	• System charges work to AMPs. • AMPState is non-idle; AMPCPU and AMPIO show resource usage.	• The system performs a small amount of work. • AMPState or PEState looks IDLE most of the time and does not show resource usage.
RESTORE	• The system performs a small amount of work. • AMPState or PEState looks IDLE most of the time and does not show resource usage.	• System charges work to AMPs. • AMPState is non-idle; AMPCPU and AMPIO show resource usage.	• The system performs some of work. • AMPState is non-idle; AMPCPU and AMPIO show resource usage.

ARC Commands	One SQL Session	*n* HUTPARSE Sessions	One HUTCTL Session
ROLLFORWARD/ ROLLBACK	• The system performs a small amount of work. • AMPState or PEState looks IDLE most of the time and does not show resource usage.		• System charges work to AMPs. • AMPState is non-idle; AMPCPU and AMPIO show resource usage.

Session Partitions Used

An ARC job uses one of three different session partitions, depending on whether the system performs an ARCHIVE or RESTORE, or ROLLFORWARD or ROLLBACK operation.

Partition	Description
Teradata SQL	Typically used in a manner similar to the Teradata SQL partition in FastLoad/MultiLoad. This session includes activity such as parsing, security checking, and controlling job execution. PEs mainly perform the work in this partition.
HUTPARSE	Similar to the FastLoad partition. Within this partition, the same job has multiple sessions, which use AMP resources to transfer data during an Archive/Restore operation. AMPs mainly perform the work in this partition.
HUTCTL	Special variation of the HUTPARSE partition. The system uses this partition when it does not require parallel execution of multiple sessions. The system uses only one HUTCTL session during an ARC job for operations such as rollback, rollforward, or build. AMPs mainly perform the work in this partition.

Teradata Manager and ARC

As a general rule, ARC operations result in two classes of commands, and the class of command determines which partitions become involved.

Command Class	Description
Low data volume transfer	The system initially processes ROLLFORWARD, ROLLBACK, and BUILD operations during the Teradata SQL session. Once the setup for the operation has completed, the system performs the actual work involved in the operation during the session associated with the HUTCTL Partition.

Command Class	Description
High data volume transfer	Commands such as ARCHIVE and RESTORE use a number of sessions to maximize the volume of data the system can transfer between the host (or client) and Teradata Database in the minimum time. The system performs setup during the Teradata SQL session, and performs real work in the other partitions. HUTPARSE sessions transfer data for the Archive/Restore operation. The system uses the HUTCTL session for portions of the process not related to Teradata Database to and from host (or client) data transfer.

Some possibly unusual items might show up while monitoring sessions:

- Not all partitions are actually used in all jobs.
- Even during the data transfer portion of an ARCHIVE operation, the system may not use all HUTPARSE sessions. If the number of HUTPARSE sessions is greater than the number of AMPs, the number of sessions actually used is a multiple of the number of AMPs. Other available HUTPARSE sessions remain idle.
- Neither HUTPARSE nor HUTCTL sessions have a defined RunProcId (that is, the vprocs that execute the work initiated by these sessions change from request to request).

The main thing you might notice while monitoring sessions with Teradata Manager is that ARC may not make even use of available AMP resources.

In the following situations, DiskReads, DiskWrites, CPUUse, and DiskUse are noticeably higher on a subset of the AMPs than on other AMPs.

Situation	Description
Specific AMP journal archive or restore	Journal ARCHIVE or RESTORE operations that involve specific AMPs impact a subset of the vprocs. Note that usage statistics may show AMPs (other than those indicated in the Utility Command) have activity. This is true when the Permanent Journal (PJ) contains single after images; the system stores single after images on an AMP in the cluster designated as backup. Since all single afterimages for a given AMP are stored on a single backup AMP, the backup AMPs show measurable usage.
Cluster level archive or restore	If the archive or restore operation takes advantage of the ability to perform ARC at the cluster level, the resources consumed during that operation are almost entirely within that cluster.
All-AMP archive performed with less sessions than AMPs	During archive operations, each HUTPARSE session is associated with an AMP until the AMP archives an entire subtable worth of data. If there are fewer sessions than AMPs, resource consumption tends to move from AMP to AMP until all AMPs have archived their portion of the table. The process starts over again on the next subtable to be archived.

Situation	Description
Restore from an archive performed with less sessions than AMPs	If the system performs a archive operation with fewer sessions than AMPs, resource usage is skewed. When resource usage is skewed, the system restores data to the AMPS unevenly.

As an example, the system uses only one HUTPARSE session in a archive operation. Given either a large table or a fast ResUsage sampling rate, you could watch resource usage move from AMP to AMP as the HUTPARSE session transfers data from an AMP to the host (or client), one AMP at a time.

During the restore operation, there is no direct tie between sessions and AMPs. Any session transfers data to whatever AMP on which it is to be stored.

Since the system writes data to the archive tape one AMP at a time, the system bombards each AMP with data until that AMP is restored, then starts loading data on the next AMP. You could watch resource usage move from AMP to AMP as the system transfers data during a HUTPARSE sessions from the host (or client) to an AMP, in some AMP order.

Given an LSN and user name, the system can accumulate all work a job performs as a total that may supply the most useful level of information.

Monitoring HUT Locks

During an Archive/Recovery operation, the utility places locks on the objects affected by the operation. These locks remain active during a Teradata Database restart, and must be explicitly released by the RELEASE LOCK command or by the RELEASE LOCK option on the ARCHIVE, ROLLBACK, ROLLFORWARD, RESTORE, and BUILD commands.

When monitoring Host Utility (HUT) locks placed on a database or table, Teradata Manager reports the object that is locked, but not necessarily who is causing the lock to block (blocker).

This occurs because HUT locks remain until they are explicitly removed, although the session holding the HUT lock may not be active. For example, the session may be logged off or aborted.

In this case, Teradata Manager does not tell you who is causing the lock and returns the Blk_x_HostId, Blk_x_SessNo, and Blk_x_UserId fields as NULLs. Use the ShowLocks utility to obtain the name of the user that placed the HUT lock. For more information on ShowLocks, *Utilities*.

Releasing HUT Locks with RELEASE LOCK

The RELEASE LOCK command removes HUT locks from the identified databases or tables. If the locks were placed at the database level, then they must be released at the database level.To release HUT locks, log on to an ARC session and submit the RELEASE LOCK command.You must have either the ARCHIVE or the RESTORE privilege on or be the owner of the database or table for which the locks are to be released.

For more information on HUT locks and the RELEASE LOCK command, see *Teradata Archive/Recovery Utility Reference*.

Monitoring a FastExport Job

This section describes how the nature of work done during the data export phase from the Teradata Database server to the client determines where and when the work is charged to the different sessions. Consequently, the pattern of work charged to different sessions causes Teradata Manager to display resource usage of data in a particular way.

Logon Sequence Number (LSN)

In running a FastExport job, a user is logged onto two Teradata SQL sessions and also, under the same user name, *n* number of EXPORT sessions. The system logs EXPORT session partitions under the same user name and under the same LSN as the Teradata SQL session partition. An LSN is associated with each session when the session logs on and identifies a collection of sessions performing the same job. To see the total impact of a FastExport job, calculate the total impact of all sessions with the same LSN.

Resource Usage

For those sessions in which work is charged to AMPs, Teradata Manager displays:

- AMPState is non-idle.

  This typically means that AMPState may be ACTIVE or BLOCKED.
- PEState in EXPORT sessions is UNKNOWN.
- AMPCPUSec and AMPIO (logical I/O) show resource usage.

The following table describes the three primary phases of a FastExport job and shows where resource usage occurs.

- In the data selection phase, the system processes the Teradata SQL SELECT request to retrieve a large amount of data from the Teradata Database server and to place the data into a spool file. The system performs most of the work on the AMPs in the Request SQL session. The minimal PE work done involves parsing.
- In the distribution phase, the system prepares data for return to the client. It is difficult to determine when this phase starts because no message is displayed. However, when the distribution phase completes, the `Select execution completed` message displays. In this phase, the system performs all work on the AMPs in the Request SQL session.
- In the data transfer phase, the AMPs return data to the client over multiple sessions. The system performs work during the *n* EXPORT sessions and charges that work to the AMPs. The system reports AMP resource usage for those EXPORT sessions, and reports the PEState as unknown.

FastExport Phases	Start of FastExport Phase	Request SQL Session	*n* EXPORT Sessions
Data selection	Select request submitted to the DBC	**1** The system charges work to AMPs. **2** In Teradata Manager, AMPState is ACTIVE; AMPCPU and AMPIO show resource usage.	
Distribution		**1** The system charges work to AMPs. **2** AMPCPU and AMPIO show resource usage.	
Data transfer	Select execution completed		**1** The system charges work to AMPs. **2** AMPState or PEState is non-idle most of time; AMPCPU and AMPIO show resource usage

The FastExport client utility uses Log SQL session (not shown) to read the restart log in the event of a system restart so that the utility knows when to recover the job. Because the work performed is minimal, this session does not show resource usage.

Partitions Used

The system performs work in two different partitions during the data transfer phase of a FastExport job.

Partition	Description
Teradata SQL	The system logs on two sessions under this partition.
EXPORT	The system logs on *n* sessions under this partition. The value of *n* is calculated as greater than one and less than or equal to the number of AMPs. Each session has a RunVProcNo associated with an AMP.

APPENDIX C Error Logs

This appendix describes the various types of event logs and error logs you can use to identify problems on the system, including:

- Types of logs and their contents, including:
  - Tables of node error logs

    The node error logs are common to all Teradata Database installations.
  - Administration Workstation (AWS) error logs

    The AWS logs are applicable only to Massive Parallel Processing (MPP) systems.
- Procedures for accessing some of these logs
- Information about verifying the BYNET
- Viewing the SW_Event_Log

Note: You can view Microsoft Windows error logs using Windows Event Viewer. For more information on these logs and using the event viewer, consult your Windows help documentation.

Log Error Table for Bulk Loading

You can use the CREATE ERROR TABLE SQL statement to create an error table for bulk data loading operations such as INSERT…SELECT and MERGE minibatches.

There are several restrictions when using a log table:

- You can only define one error table per data table.
- Error tables have the same fallback properties as their associated data tables.
- Error tables have the same PI as their associated data tables.
- You must create an error table before you can log errors.
- You cannot define triggers, JIs, or HIs on an error table.

You must correct the errors manually.

For more information on the CREATE ERROR TABLE syntax, a complete list of restrictions, and usage rules, see *SQL Reference: Data Definition Statements*.

Log File Summary

This section contains node and AWS error log information.

Note: The Cabinet Management Interface Controller (CMIC) Garbage Flag filters the bulk of CMIC messages and prevents them from being collected in the Console Log. Turn this flag ON for each cabinet.

Teradata Database and Kernel Error Sequence

On MP-RAS, the system must pass messages for Teradata Database errors through the logger daemon before inserting the errors into the log. Sometimes Teradata Database errors can appear after error messages logged by the kernel, even though the Teradata Database error actually occurred first.

This is particularly true when a Trusted Parallel Application (TPA) reset occurs. You usually see messages logged by kernel reset code in the log prior to the message for the event that initiated the reset.

Node Error Logs

The following table describes the error logs for nodes, where they reside, how long they persist, and what the contents contain.

This directory/file name ...	Contains ...	And lasts ...
/etc/.osm	• A copy of most of the console output information. All console output information is stored in the */var/console/console.log* file on the AWS. • Output from the reconcile utility.	until the next reboot. During a reboot, the file */etc/.osm* gets moved to */etc/.osm.old*. Upon a second reboot, this information is lost.
*/var/adm/streams/**	files for each day of the year, uniquely identified with a month and day. Each file includes almost all system error, BYNET error, and PDE error.	no more than two weeks. A UNIX cron job runs every Sunday night just after midnight, which renames all *error** files to *oerror** and deletes the old set of *oerror* files. If you disable the cron job, files accumulate until they fill up */var*. If storage is available, the date-unique filenames make these files good for one year.
/tpi-data/nodecheck.tpacycle_*n* (UNIX) …\tdConfig\tpi-data\tpacycle_*n* (Windows) where: *n* is the count extracted from the running PDE kernel memory	entries generated by the Resource Check Tools nodecheck utility (if you ran the nodecheck command with the -L argument). If file number *n* already exists, the previous version is appended with .old to save up to one level of history logs.	until you remove them with the syscheckrm command-line utility. If you enter the command `syscheckrm` without arguments, the utility prints the list of log files in all tpa nodes and removes any corresponding .old files.

This directory/file name …	Contains …	And lasts …
*/var/array/logs/** (LSI only)	a file for each disk array that the node accesses. Each file includes all errors detected for that disk array. When an error exists in any file, that error is displayed once per hour to the console of that node until the error is removed from the file. To remove the error, enter: `# /opt/adpxspt/clearlogs`	as long as you want. There is no built-in scheme for management of these files. You must manage them manually.
*/var/adm/usererror/**	log files with error information concerning: • Saving Teradata Database crashdumps • DMP errors • csp errors • Gateway errors These logs are similar to the */var/adm/streams/* logs, except that they typically contain messages logged by user-level daemons. Most PDE and DBS errors go to the */var/adm/streams/* log, even those logged by user-level programs.	no more than two weeks. A standard UNIX cron job runs every Sunday night just after midnight, which renames all *error** files to *oerror** and deletes the old set of *oerror* files. If you disable the cron job, files accumulate until they fill up */var*. If storage is available, the date-unique filenames make these files good for one year.
/sdd/logs/restart.log	a copy of the *xpt* output from its shutdown and its startup.	until the next shutdown. The system creates or empties this file during shutdown, then duplicates all output of xptinit in the file. The file is cumulative until the next system shutdown. At startup, the system empties shutdown information from this file and then writes startup information into this file.
/tdsw/recon_out	the output from the last reconcile performed on the system. You should view it only on the Package Distribution Node (PDN) or on the node where the reconcile occurred.	until the next reconcile. This file is overwritten at the start of a new reconcile. You can view previous reconcile output in the */etc/.osm* file.
• */tdsw/logtms/tmslog* • */tdsw/logtms/tmsdlog*	log files exist for proc daemons. Server *tmssrvr* log files exist on all nodes. The daemon *tmsd* log file is only on the PDN.	until the next reboot of a node. At a node reboot: **1** The system renames the current log files with a *.old* extension in this directory. **2** The system starts new log files start for *tmsd* and *tmssrvr*.

AWS Error Logs

These error logs are generated by the AWS administering an MPP system.

This directory/file name ...	Contains ...	And lasts ...
*/var/adm/streams/**	files for each day of the year, uniquely identified with a month and day. Each file includes all AWS system error information.	no more than two weeks. A standard UNIX cron job runs every Sunday night just after midnight, which renames all *error** files to *oerror** and deletes the old set of *oerror* files. **Note:** If you disable the cron job, files accumulate until they fill up */var*. If storage is available, the date-unique filenames make these files good for one year.
/var/console/console.log	a mirror of all information that goes to */dev/console/* of all nodes that are managed by this AWS. You can display this file via the AWS GUI interface using the Status >Event menu selections.	until the */var/console/console.log* file grows to 500K in size. Then the system moves the file contents to a uniquely stamped file in the */var/console/old_logs/ directory.* The amount of information saved depends on: • The number of nodes managed by this AWS • The frequency of information transfer to */dev/consoles* of all the nodes • The size of the */var* file system on the AWS
CSF event list	all of the errors, warnings, and state changes that occur on a 5100 machine. This list has its own GUI interface.	indefinitely. **Note:** It is up to you to remove entries in the list.
DBC.software_event_log	the Teradata Database view of software events.	indefinitely. **Note:** It is up to you to remove entries.

Viewing Log Files

This section provides procedures for viewing the console log and node logs.

Log File Error Formats

Both the */var/adm/streams/* and */var/adm/usererror/* logs contain a standard header at the beginning of each line.

- The first column is a sequential message number, reset to 1 at boot time.
- The next column contains the timestamp of when the message is logged.
- The timestamp is followed by 70-80 characters that contain encoded header information which is not usually very useful.
- The actual message text appears at the end of the line.

Some messages are too long to fit onto one line, and are continued on following lines. You can tell which lines are continuations because they do not contain a header.

Displaying Log Files

You can use the errpt UNIX utility to display the */var/adm/streams/* and */var/adm/usererror/* logs. The errpt utility decodes the header information in each entry, and some kinds of encoded entries.

Viewing the Console Log

The AWS Consolidated Console Log brings together log messages from every node. To view the Consolidated Console Log on the AWS:

1. Bring up the main AWS window.
2. From the Status menu, select Logs > Console Logs.

   The Console Log window displays the UNIX console output.
3. To save the console output, choose Save Log from the File menu.
4. To exit the Console log window, choose *Exit* from the File menu.

Viewing Node Logs

To view the log files for a particular node, perform the following steps in a UNIX shell on that node:

1. To access a node log file, access a UNIX shell on the node and go to the directory:

   ```
   # cd /var/adm/streams
   ```
2. To see what files are there, enter:

   ```
   # ls -l
   ```
3. To examine a particular log file, for example, *error.10-05*, enter:

   ```
   # /usr/bin/errpt /var/adm/streams/error.10-05 >
   /var/adm/streams/error.10-05.errpt
   ```
   The command creates an a readable ASCII file named *error.10-05.errpt*.
4. To examine a particular log file that has information about disk arrays (for example, *error.10-07*), enter:

   ```
   # /opt/adpxspt/errlog /var/adm/streams/error.10-07 > /var/adm/
   streams/error.10-07.errlog
   ```
   The command creates an a readable ASCII file named *error.10-07.errlog*.

   You may find the log files from the following directories useful for debugging:

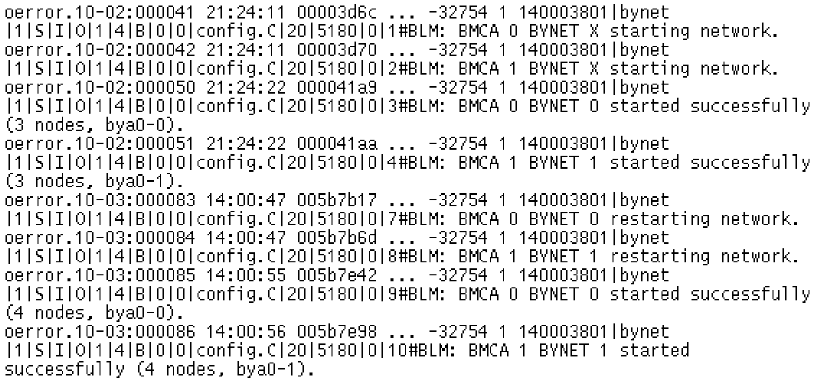
   - */var/adm/usererr*
   - */var/array/logs*
   - */var/adm/streams*
   - */etc/.osm and /etc/.osm.old*
   - */var/console/console.log*

Viewing BYNET Activity

To verify if BYNET is running successfully on each node after a system reboot, enter the following command, using the appropriate month and day numbers:

```
# grep BMCA /var/adm/streams/error.mm-dd | tail
```

Messages similar to the following appear (where BLM means BYNET Link Manager):



```
oerror.10-02:000041 21:24:11 00003d6c ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|1#BLM: BMCA 0 BYNET X starting network.
oerror.10-02:000042 21:24:11 00003d70 ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|2#BLM: BMCA 1 BYNET X starting network.
oerror.10-02:000050 21:24:22 000041a9 ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|3#BLM: BMCA 0 BYNET 0 started successfully
(3 nodes, bya0-0).
oerror.10-02:000051 21:24:22 000041aa ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|4#BLM: BMCA 1 BYNET 1 started successfully
(3 nodes, bya0-1).
oerror.10-03:000083 14:00:47 005b7b17 ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|7#BLM: BMCA 0 BYNET 0 restarting network.
oerror.10-03:000084 14:00:47 005b7b6d ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|8#BLM: BMCA 1 BYNET 1 restarting network.
oerror.10-03:000085 14:00:55 005b7e42 ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|9#BLM: BMCA 0 BYNET 0 started successfully
(4 nodes, bya0-0).
oerror.10-03:000086 14:00:56 005b7e98 ... -32754 1 140003801|bynet
|1|S|I|O|1|4|B|0|0|config.C|20|5180|0|10#BLM: BMCA 1 BYNET 1 started
successfully (4 nodes, bya0-1).
```

Note: BLM is an acronym for BYNET Link Manager. Examine the last few lines (BLM messages 9# and 10#) of the example. In this instance, BYNET 0 and BYNET 1 started successfully.

Viewing the Software_Event_Log

Teradata recommends that you access the system view DBC.Software_Event_Log rather than the underlying table DBC.SW_Event_Log. You can view the contents of the log by submitting the query:

```
SELECT * FROM DBC.Software_Event_Log ;
```

For Software_Event_Log column names and formats, see *Data Dictionary*.

The system views should already have been created in DBC during the Teradata Database installation process. If you need to create them now, see “Database Initialization Program (DIP) Utility” in *Utilities*.

Glossary

2PC Two-Phase Commit

ACI Applicant Coordinator Interface

AMP Access Module Process

ANSI American National Standards Institute

API Applicant Participant Interface

ARC Archive and Recovery

ASE Account String Expansion

AWS Administration Workstation

BLM BYNET Link Manager

BLOB Binary Large Object

BTEQ Basic Teradata Query

BYNET Banyan Network

CICS Customer Information Control System

CRJ Changed Row Journal

CLI Call Level Interface

CLIv2 Call Level Interface Version 2

CLOB Character Large Object

CMIC Cabinet Management Interface Controller

COBOL Common Business Oriented Language

CPI Coordinator Participant Interface

CPU Central Processing Unit

CR Cylinder Read

CSP Copy Save Program

DAC Disk Array Controller

DBQL Database Query Logging

DBW Database Window

DCL Data Control Language

DD Data Dictionary

DDL Data Definition Language

DIP Database Initialization Program

DL/I Data Language/I - the IMS data manipulation language

DML Data Manipulation Language

DMP Dump Memory Program

DSS Decision Support System

DSU Data Storage Unit

DUL Dump Unload/Load

DULTAPE Dump Unload/Load Tape

DWM Teradata Dynamic Workload Manager

FDL Fast Data Load, FastLoad

FLoad FastLoad Utility

FIFO First In First Out

FK Foreign Key

FSG File System Segment

GDO Globally Distributed Object

GSS Generic Security Services

HCA Host Channel Adapters

HI Hash Index

HUT Host Utility (in particular, ARC)

HUTCNS Host Utility Console Subsystem

ID Identity Column

nameID Identifier

IMS Information Management System

I/O Input/Output

JDBC Java Database Connectivity

JAR Java Archive file. A file format that contains multiple files and is used to distribute a complete Java application.

JI Join Index

KRB5 Kerberos

LAN Local Area Network

LDAP Lightweight Directory Access Protocol

LOB Large Object

LSN Logon Sequence Number

LUN Logical Unit

MLoad MultiLoad

MOSI Micro Operating System Interface

MPP Massively Parallel Processing

MTDP Micro Teradata Director Program

NPPI Nonpartitioned Primary Index

NTLM Windows NT Lan Manager

NUPI Nonunique Primary Index

NUPPI Nonunique Partitioned Primary Index

NUSI Nonunique Secondary Index

ODBC Open Database Connectivity

OJ Ordered System Change Journal

OLE DB Object Linking and Embedding for Databases

OLTP Online Transaction Processing

OpEnv Operating Environment (Teradata Dynamic Workload Manager concept)

OS Operating system

OSCJ Ordered System Change Journal

OTB Open Teradata Backup solution

PCI Participant Coordinator Interface

PDE Parallel Database Extensions

PDN Package Distribution Node

PE Parsing Engine

PI Primary Index

PJ Permanent Journal

PK Primary Key

PL/I Programming Language/1

PM/API Performance Monitor Application Programming Interface

PPI Partitioned Primary Index

PSA Priority Scheduler Administrator

QCD Query Capture Database

QCF Query Capture Facility

RAID Redundant Array of Independent Disks

RCC Recovery Control Catalog

RCT Resource Check Tools

RDAC Redundant Disk Array Controller

RDBMS Relational Database Management System

RI Referential Integrity or Referential Index

RSG Relay Services Gateway vproc

SCSI Small Computer System Interface

SHA Secure Hash Algorithm

SI Secondary Index

SMP Symmetric Multi-Processing

SSO Single Sign-On

SUS Startup Subsystem

SysCon System Condition (Teradata Dynamic Workload Manager concept)

TDGSS Teradata Generic Security Services

TDP Teradata Director Program

TDQM Teradata Dynamic Query Manager (now Teradata Dynamic Workload Manager)

TWA Teradata Workload Analyzer

TWMA Teradata Workload Management Administrator

TJ Transient Journal

TLE Target-Level Emulation

TPA Trusted Parallel Application (always a Teradata Database)

TPT Teradata Parallel Transporter

TPump Teradata Parallel Data Pump

TSC Teradata Support Center (Global Support Center)

TSET Teradata System Emulation Tool

TUVT Teradata Utilities Verification Tool

UDF User-Defined Function

UDT User-defined Type

UPI Unique Primary Index

UPPI Unique Partitioned Primary Index

USI Unique Secondary Index

VE Visual Explain

VEComp Visual Explain and Compare

vdisk Virtual Disk

vproc Virtual Processor (an AMP or PE)

VSAM Virtual Sequential Access Method

WD Workload Definition (Teradata Dynamic Workload Manager concept)

Index