



Teradata Factory

Student Binder #2

Course # 9038

Revision 12.0.0



Student Guide

Teradata Training

Notes

Table of Contents

Module 30 – Additional Index Choices

Additional Index Choices.....	30-4
Join Indexes.....	30-6
Options for Join Indexes	30-8
Join Index Considerations.....	30-10
Join Index Example – Customer and Order Tables	30-12
Compressed Multi-Table Join Index.....	30-14
Non-Compressed Multi-Table Join Index.....	30-16
Example 1 – Does a Join Index Help?	30-18
Example 2 – Does a Join Index Help?	30-20
Join Index – Single Table.....	30-22
Join Index – Single Table (cont.)	30-24
Creating a Join Index – Single Table	30-26
Example 3 – Does the Join Index Help?	30-28
Why use Aggregate Join Indexes?	30-30
Aggregate Join Index Properties	30-32
Aggregation without an Aggregate Index	30-34
Creating an Aggregate Join Index.....	30-36
Aggregation with an Aggregate Index	30-38
Sparse Join Indexes	30-40
Creating a Sparse Join Index.....	30-42
Creating a Sparse Join Index on Partitions	30-44
Global (Join) Indexes	30-46
Global Index – Multiple Tables	30-48
Global Index as a “Hashed NUSI”.....	30-50
Creating a Global Index (“Hashed NUSI”).....	30-52
Example: Using a Global Index as a Hashed NUSI.....	30-54
Repeating Row Ids in Global Index	30-56
Hash Indexes	30-58
Creating a Hash Index.....	30-60
Hash Index – Example 2	30-62
Using a Hash Index for Joins – Example 3	30-64
Hash Index – Example 3 (cont.).....	30-66
Summary	30-68
Review Questions	30-70
Lab Exercise 30-1	30-72

Module 31 – Miscellaneous SQL Features

Teradata SQL.....	31-4
SQL – Version 2 Differences	31-6
SQL Session Modes.....	31-8
Transaction Modes – Teradata.....	31-10
Transaction Modes – ANSI	31-12
Transaction Mode Examples	31-14
Multi-Statement Requests.....	31-16
CASE Sensitivity Issues	31-18
Setting the SQL Flagger	31-20
SQLFLAG Example	31-22
HELP SESSION Command	31-24
Why a System Calendar?.....	31-26
Calendar Table Layout	31-28
One Row in the Calendar.....	31-30
Using the Calendar	31-32
What is a Trigger?	31-34
Trigger Example	31-36
Temporary Table Choices.....	31-40
Derived Tables Revisited.....	31-42
Volatile Tables.....	31-44
Volatile Table Restrictions	31-46
Global Temporary Tables	31-48
Creating Global Temporary Tables	31-50
V2R5 – New Features.....	31-52
V2R6 – New Features.....	31-54
Teradata Limits (Different Releases).....	31-56
Review Questions	31-58

Module 32 – Introduction to Application Utilities

Application Utilities	32-4
Application Utilities Environments	32-6
Application Development	32-8
Transferring Large Amounts of Data.....	32-10
INSERT/SELECT: The Fast Path.....	32-12
Multi-Statement Insert/Select Example	32-14
DELETE (ALL): The Fast Path	32-16
Using an INMOD Routine	32-18
Teradata Parallel Transporter.....	32-20
Teradata Parallel Transporter Operators	32-22
Referential Integrity and Load Utility Issues	32-24
Maximum Number of Load Jobs	32-26
Maximum Number of Load Jobs (cont.).....	32-28
Application Utility Checklist	32-30
Application Utility Summary	32-32
Review Questions	32-34

Module 33 – BTEQ

BTEQ	33-4
Using BTEQ Conditional Logic	33-6
BTEQ Error Handling.....	33-8
BTEQ EXPORT – Example 1	33-10
4 Types of BTEQ .EXPORT	33-12
BTEQ Data Modes	33-14
BTEQ EXPORT – Example 2	33-16
BTEQ EXPORT – Example 3	33-18
Indicator Variables.....	33-20
Determining the Logical Record Length with Fixed Length Columns	33-22
Determining the Logical Record Length with Variable Length Columns	33-24
Determining the Logical Record Length with .EXPORT INDICDATA	33-26
.IMPORT (for Network-Attached Systems).....	33-28
.IMPORT (for Channel-Attached Systems)	33-30
.PACK.....	33-32
.REPEAT	33-32
BTEQ IMPORT – Example 1.....	33-34
BTEQ IMPORT – Example 2.....	33-36
BTEQ IMPORT – Example 3.....	33-38
Multiple Sessions.....	33-40
.SET SESSIONS	33-42
Parallel Processing Using Multiple Sessions to Access Individual Rows.....	33-44
When Do Multiple Sessions Make Sense?	33-46
Application Utility Checklist.....	33-48
Review Questions	33-50
Lab Exercise 33-1	33-52
Lab Exercise 33-2	33-56

Module 34 – FastLoad

FastLoad	34-4
FastLoad Characteristics	34-6
FastLoad Phase 1	34-8
FastLoad Phase 2	34-10
A Sample FastLoad Script	34-12
Converting the Data	34-14
Data Conversion Chart.....	34-16
NULLIF	34-18
FastLoading Zoned Decimals and Time Stamps	34-20
FastLoad BEGIN LOADING Statement	34-22
BEGIN LOADING Statement	34-22
FastLoad Error Tables.....	34-24
Error Recovery	34-26
CHECKPOINT Option	34-28
END LOADING Statement	34-30
RECORD Statement	34-32
INSERT Statement.....	34-34
Staged Loading of Multiple Data Files	34-36
FastLoad Fails to Complete	34-38
Restarting FastLoad (Output).....	34-40
Restarting FastLoad – Summary	34-42
Additional FastLoad Commands	34-44
FastLoad with Additional Options	34-46
Invoking FastLoad	34-48
INMOD	34-50
Application Utility Checklist	34-52
Summary	34-54
Review Questions	34-56
Lab Exercise 34-1	34-58
Lab Exercise 34-2	34-60

Module 35 – The Support Environment

Support Environment.....	35-4
Setting Up the Support Environment.....	35-6
Invoking Utilities	35-8
Support Environment Commands.....	35-10
Initializing the Log Table	35-12
Initialization and Wrap Up Commands	35-14
.ACCEPT – Working with Variables	35-16
Support Environment System Variables.....	35-18
.DISPLAY and .ROUTE Commands	35-20
Example: Using Variables in a Script.....	35-22
Working with Control Logic	35-24
Support Environment Example – Input	35-26
Support Environment Example – Output	35-28
Teradata SQL Support	35-30
Script – Example Input	35-32
Script – Example Output	35-34
Summary	35-36
Review Questions	35-38
Lab Exercise 35-1	35-40

Module 36 – FastExport

FastExport.....	36-4
.BEGIN and .END EXPORT.....	36-6
.EXPORT	36-8
A FastExport Script	36-10
The SELECT Request.....	36-12
Impact of Requesting Sorted Output	36-14
The SORT Procedure.....	36-16
Multiple Exports in one FastExport Job	36-18
Invoking FastExport	36-20
FastExport and Variable Input.....	36-22
A FastExport Script with ACCEPT	36-24
A FastExport Script with LAYOUT	36-26
.LAYOUT, .FIELD, and .FILLER	36-28
INMODs and OUTMODs	36-30
OUTMOD Return Codes.....	36-32
Application Utility Checklist.....	36-34
Summary	36-36
Review Questions	36-38
Lab Exercise 36-1	36-40
Lab Exercise 36-2	36-42

Module 37 – MultiLoad Part 1

What is MultiLoad?	37-4
MultiLoad Limitations	37-6
How MultiLoad Works	37-8
Advantages of MultiLoad	37-10
Basic MultiLoad Statements	37-12
Sample MultiLoad IMPORT Task.....	37-14
IMPORT Task.....	37-16
5 Phases of IMPORT Task.....	37-18
Phase 1: Preliminary	37-20
Phase 2: DML Transaction	37-22
Phase 3: Acquisition.....	37-24
Phase 3: Acquisition – a Closer Look	37-26
Phase 4: Application	37-28
Phase 4: Application – a Closer Look.....	37-30
Phase 5: Cleanup.....	37-32
Sample MultiLoad DELETE Tasks	37-34
DELETE Task Differences from IMPORT Task.....	37-36
A Closer Look at DELETE Task Application Phase.....	37-38
MultiLoad Locks.....	37-40
Restarting MultiLoad	37-42
RELEASE MLOAD Statement	37-44
Invoking MultiLoad	37-46
Application Utility Checklist	37-48
Summary	37-50
Review Questions	37-52
Lab Exercise 37-1	37-54

Module 38 – MultiLoad Part 2

New Accounts Application – Description	38-4
.BEGIN IMPORT Task Commands	38-12
Work Tables.....	38-14
Error Tables	38-16
ERRLIMIT	38-18
CHECKPOINT	38-20
More .BEGIN Parameters.....	38-22
More .BEGIN Parameters: AMPCHECK	38-24
DELETE Task Commands	38-26
.LAYOUT and .TABLE	38-28
.LAYOUT Parameters — CONTINUEIF	38-30
.LAYOUT Parameters — INDICATORS.....	38-32
.FIELD and .FILLER.....	38-34
.LAYOUT Command — Examples	38-36
Redefining the Input – Example	38-38
The .DML Command Options.....	38-40
MultiLoad Statistics.....	38-44
INMOD.....	38-46
Summary.....	38-48
Review Questions	38-50
Lab Exercise 38-1	38-52
Lab Exercise 38-2	38-54

Module 39 – TPump

TPump	39-4
TPump Limitations	39-6
.BEGIN LOAD Statement	39-8
TPump Specific Parameters	39-10
.BEGIN LOAD – PACK and RATE	39-12
.BEGIN LOAD – SERIALIZE OFF	39-14
.BEGIN LOAD – SERIALIZE ON	39-16
.BEGIN LOAD – ROBUST ON	39-18
Sample TPump Script (1 of 2)	39-20
Sample TPump Script (2 of 2)	39-22
TPump Compared with MultiLoad	39-24
Additional TPump Statements	39-26
Invoking TPump	39-28
TPump Statistics	39-30
TPump Monitor	39-32
INMOD	39-34
Application Utility Checklist	39-36
Summary	39-38
Review Questions	39-40
Lab Exercise 39-1	39-42

Module 40 – Choosing a Utility

Solution 1: Update or Delete vs. Insert/Select	40-4
Solution 2: SQL Update vs. MultiLoad or TPump	40-6
Solution 3: SQL Update vs. FastLoad	40-8
Utility Considerations	40-10
Various Ways of Performing an Update	40-12
Choosing a Utility Exercise	40-14

Module 41 – Database Administration and Building the Database Environment

Database Administration	41-4
Initial Teradata Database	41-6
Administrative User.....	41-8
Owners, Parents and Children	41-10
Creating New Users and Databases	41-12
Transfer of Ownership.....	41-14
DELETE/DROP Statements	41-16
Teradata Administrator – New System.....	41-18
Teradata Administrator – Hierarchy	41-20
Summary	41-22
Review Questions	41-24

Module 42 – The Data Dictionary

Data Dictionary / Directory.....	42-4
Fallback Protected Data Dictionary Tables.....	42-6
Non-Hashed Data Dictionary Tables	42-8
Updating Data Dictionary Tables	42-10
Supplied Data Dictionary Views.....	42-12
Restricted Views	42-14
Suffix Options with Views.....	42-16
Selecting Information about Created Objects	42-18
Children View	42-20
Databases View.....	42-22
Users View	42-24
Tables View	42-26
Columns View.....	42-28
Indices View.....	42-30
IndexConstraints View.....	42-34
ShowTblChecks View.....	42-36
ShowColChecks View	42-38
Triggers View.....	42-40
AllTempTables View	42-42
Referential Integrity Views.....	42-44
Using the DBC.Tables View.....	42-46
Referential Integrity States.....	42-48
DBC.All_RI_Children View.....	42-50
DBC.Databases2 View.....	42-52
Time Stamps in Data Dictionary.....	42-54
Teradata Administrator – List Columns of a View.....	42-56
Teradata Administrator – Object Options	42-58
Summary	42-60
Review Questions	42-62
Lab Exercise 42-1	42-64
Lab Exercise 42-2	42-70

Module 43 – Space Allocation and Usage

Permanent Space Terminology	43-4
Spool Space Terminology.....	43-6
Temporary Space Terminology	43-8
Resetting Peak Values	43-10
Assigning Perm and Spool Limits	43-12
Giving One User to Another.....	43-14
Teradata Administrator – Move Space	43-16
Reserving Space for Spool.....	43-18
Views for Space Allocation Reporting	43-20
DiskSpace View.....	43-22
TableSize View.....	43-24
AllSpace View	43-26
DataBaseSpace Table	43-28
Different Views — Different Results	43-30
Additional Utilities to View Space Utilization.....	43-32
Teradata Administrator – Database Menu Options	43-34
Teradata Administrator – Object Menu Options	43-36
Transient Journal Space	43-38
Ferret Utility	43-40
Ferret SHOWSPACE Command.....	43-42
Ferret SHOWBLOCKS	43-44
Review Questions	43-48

Module 44 – Users, Accounts, and Accounting

Creating New Users & Databases.....	44-4
CREATE DATABASE Statement.....	44-6
CREATE USER Statement.....	44-8
CREATE USER and the Data Dictionary.....	44-10
CREATE USER and the Data Dictionary (cont.)	44-12
MODIFY USER Statement.....	44-14
Teradata Administrator – Tools Menu > Create Options	44-16
Creating and Using Account IDs	44-18
Dynamically Changing an Account ID	44-20
Account Priorities	44-22
Account String Expansion.....	44-24
ASE Accounting Example	44-26
System Accounting Views	44-28
DBC.AccountInfo[X] View	44-30
DBC.AMPUsage View	44-32
DBC.AMPUsage View — Examples	44-34
Users, Accounts & Accounting Summary	44-36
Review Questions	44-38
Lab Exercise 44-1	44-40

Module 45 – Access Rights

Privileges/Access Rights	45-4
Access Rights Mechanisms	45-6
CREATE TABLE – Automatic Rights.....	45-8
CREATE USER – Automatic Rights	45-10
Implicit, Automatic, and Explicit Rights.....	45-12
GRANT Command.....	45-14
Granting Rights at Database Level	45-16
GRANT Rights at the Table or Column Level.....	45-18
REVOKE Command	45-20
Revoking Non-Existent Rights	45-22
Removing a Level in the Hierarchy	45-24
Inheriting Access Rights.....	45-26
The GIVE Statement and Access Rights	45-28
Access Rights and Views.....	45-30
Access Rights and Nested Views	45-32
System Views for Access Rights	45-34
DBC.AllRights[X] and DBC.UserRights Views.....	45-36
DBC.UserGrantedRights View.....	45-38
Teradata Administrator – Grant/Revoke Rights	45-40
Teradata Administrator – Rights on DB/User	45-42
Access Rights Summary	45-44
Review Questions	45-46

Module 46 – Roles and Profiles

What are Roles and Profiles?	46-4
Advantages of Roles	46-6
Access Rights without Roles.....	46-8
Access Rights Issues (prior to Roles).....	46-8
Access Rights Using a Role	46-10
Implementing Roles	46-12
Current or Active Roles	46-14
Nesting of Roles.....	46-16
Example of Using “Nested Roles”	46-18
Access Rights Validation and Roles	46-20
SQL Statements to Support Roles.....	46-22
GRANT Command (SQL Form)	46-24
REVOKE Command (SQL Form).....	46-26
GRANT and REVOKE Commands (Role Form).....	46-28
System Hierarchy (used in following examples)	46-30
Example of Using Roles.....	46-32
Example of Using Roles (cont.)	46-34
Example of Using Roles (cont.).....	46-36
RoleInfo[X] View	46-38
RoleMembers[X] View	46-40
DBC.AccessRights and “Rights” Views.....	46-42
AllRoleRights and UserRoleRights Views	46-44
Steps to Implementing Roles	46-46
Profiles	46-48
Example of Simplifying User Management.....	46-50
Implementing Profiles	46-52
Impact of Profiles on Users.....	46-54
CREATE/MODIFY PROFILE Statement	46-56
Password Attributes (CREATE/MODIFY PROFILE)	46-58
Teradata Password Control (V2R6.1)	46-60
Teradata Password Control (V2R6.1) – cont.....	46-62
Teradata Password Control (V2R6.1) Options	46-64
CREATE PROFILE Example.....	46-66
Teradata Administrator CREATE PROFILE Example	46-68
DROP PROFILE Statement.....	46-70
ProfileInfo[X] View	46-72
Miscellaneous SQL Functions	46-74
Summary	46-76
Review Questions	46-78
Lab Exercise 46-1	46-80
Lab Exercise 46-2	46-84

Module 47 – Priority Scheduler

Levels of Workload Management	47-4
Priority Scheduler Facility	47-6
Priority Scheduler Architecture	47-8
Priority Scheduler Architecture with TDWM Workloads.....	47-10
Priority Scheduler Concepts	47-12
Resource Partitions and Performance Groups	47-14
Relative Weights.....	47-16
Performance Periods and Milestones.....	47-18
CPU Usage Limits with Priority Scheduler.....	47-20
Use of Performance Groups.....	47-22
Getting Started with Priority Scheduler.....	47-24
Schmon Utility	47-26
Schmon Example	47-28
Priority Scheduler Administrator.....	47-34
Summary	47-36
Review Questions	47-38

Module 48 – Access Control

System Access Control Levels.....	48-4
Teradata Password Encryption.....	48-6
Password Security Features	48-8
Teradata Connectivity	48-10
Sessions and Session Pools	48-12
Teradata Director Program (TDP)	48-14
TDP Exits	48-16
Host Logon Processing	48-18
Objects used in Host Logon Processing.....	48-20
GRANT/REVOKE LOGON Statements	48-22
GRANT/REVOKE LOGON Example	48-24
Session Related Views	48-26
DBC.LogonRules View	48-28
DBC.LogOnOff View	48-30
DBC.SessionInfo View	48-32
Additional Utilities to View Sessions	48-34
Query Session Utility	48-36
Gateway Global Utility	48-38
Access Control Mechanisms	48-40
Using Views to Limit Access.....	48-42
Using Macros to Reduce User SQL Complexity	48-44
Structure the System	48-46
A Recommended Access Rights Structure	48-48
A Recommended System Hierarchy	48-50
System Access Controls Summary	48-52
Review Questions	48-54

Module 49 – Access and Query Logging

Access and Query Logging.....	49-4
Access Logging	49-6
Objects used in Access Logging.....	49-8
BEGIN LOGGING Statement.....	49-10
END LOGGING Statement	49-12
Setting up Access Logging	49-14
Access Log Views	49-16
DBC.AccLogRules View	49-18
BEGIN LOGGING – Example.....	49-20
DBC.AccessLog View.....	49-22
DBC.AccessLog View – Example	49-24
END LOGGING – Example.....	49-26
Teradata Administrator – Tools Menu > Access Logging	49-28
Query Logging (DBQL) Concepts	49-30
Objects used in Defining Rules for DBQL.....	49-32
Objects used in DBQL (cont.)	49-34
BEGIN QUERY LOGGING Statement	49-36
BEGIN QUERY LOGGING WITH ... (cont.)	49-38
BEGIN QUERY LOGGING LIMIT ... (cont.)	49-40
BEGIN QUERY LOGGING Examples	49-42
BEGIN QUERY LOGGING Examples (cont.).....	49-44
BEGIN QUERY LOGGING Examples (cont.).....	49-46
END QUERY LOGGING Statement	49-48
DBC.DBQLRules View	49-50
DBC.QryLog View – Example.....	49-52
DBC.QryLogSummary View – Example	49-54
Teradata Administrator – Tools Menu > Query Logging.....	49-56
Access and Query Logging Summary	49-58
Review Questions	49-60
Lab Exercise 49-1	49-62
Lab Exercise 49-2	49-66

Module 50 – Workload Management

Levels of Workload Management.....	50-4
What is TASM?	50-6
Teradata Dynamic Workload Manager (TDWM).....	50-8
TDWM Query Management Architecture	50-10
Query Management Architecture (cont.)	50-12
TDWM Main Window – Types of Rules.....	50-14
Filters and Throttles for Query Management.....	50-16
Object Access and Query Resource Filters.....	50-18
Object and Load Utility Throttles	50-20
TDWM – General Settings	50-22
Object Access Filter Properties.....	50-24
Query Resource Filter Properties	50-26
Object Throttling Properties.....	50-28
Linking Objects to Rules.....	50-30
Workload Definitions.....	50-32
Creating Workloads	50-34
Example of Using Workloads	50-36
TDWM – Create Workload Definitions.....	50-38
WD – Classification Criteria.....	50-40
WD – Exception Criteria.....	50-42
TDWM – Specify Exception Criteria	50-44
Example – Exception Handling	50-46
TDWM – Priority Scheduler.....	50-50
Teradata Workload Analyzer	50-52
Teradata Query Scheduler.....	50-54
Summary	50-56
Review Questions	50-58

Module 51 – Teradata Manager and Performance Monitor

Teradata Manager	51-4
Getting Started	51-6
Teradata Manager Main Window	51-8
Teradata Manager Applications.....	51-10
Teradata Manager Dashboard.....	51-12
Teradata Performance Monitor.....	51-14
Teradata Performance Monitor Chart Function.....	51-16
Teradata Performance Monitor – Resource Usage	51-18
Teradata Performance Monitor – Resource Usage Detail	51-20
Teradata Performance Monitor – Session Information	51-22
Teradata Performance Monitor – User Session Info	51-24
Teradata Performance Monitor – SQL & Explain Steps	51-26
Teradata Manager – Database Console	51-28
Database Console Example	51-30
Teradata Manager – Locking Logger	51-32
Summary.....	51-34
Review Questions	51-36

Module 52 – Performance Monitoring

Performance Monitoring Tools	52-4
Why Collect Resource Usage Data?	52-6
Resource Usage Data	52-8
Collection Costs	52-8
Filling the ResUsage Tables.....	52-10
Specifying ResUsage Tables and Logging Rates.....	52-12
Resource Usage Tables	52-14
Resource Usage Views.....	52-16
Resource Usage Macros	52-18
Example Output from DBC.ResNode Macro	52-20
PM/API and Performance Monitor	52-22
Teradata System Emulation Tool (Teradata SET)	52-24
Performance Monitoring Summary	52-26
Review Questions	52-28

Module 53 – Utilities Overview and Teradata DB Window

Locations from which Utilities can be Executed	53-4
SMP and Database Window Utilities.....	53-6
Teradata Console Task.....	53-8
Teradata Database Window	53-10
DBW Supervisor Window	53-12
Teradata MultiTool (Windows and Linux)	53-14
DB Window via MultiTool	53-16
Accessing Console Utilities via the DB Window	53-18
DBS Control Utility	53-20
DBS Control Record – General Fields (V2R5.1)	53-22
DBS Control Record – General Fields (TD 12.0).....	53-24
DBS Control Record – File System Fields	53-26
DBS Control Record – Performance Fields	53-28
DBS Control Record – Checksum Fields.....	53-30
Modifying DBS Control Parameters.....	53-32
Summary	53-34
Review Questions	53-36

Module 54 – System Restarts

Types of Restarts	54-4
Scheduled Restarts.....	54-6
Restarting Teradata from DB Window.....	54-8
Restart using the “tpareset” Command.....	54-10
Restart Messages and Information.....	54-12
PDE States	54-14
Unscheduled Restarts.....	54-16
Unscheduled Restarts (cont.).....	54-18
Unscheduled Restarts (cont.).....	54-20
TPA Reset – Crashdumps	54-22
Allocating Crashdumps Space.....	54-24
TPA Dump Maintenance	54-26
UNIX Panic Dumps	54-26
Review Questions	54-28

Module 55 – Maintenance and Recovery Utilities

Maintenance, Diagnostic, and Recovery Utilities	55-4
Abort Host Utility	55-4
Ferret – Defragment and Packdisk	55-6
Checking Data Integrity.....	55-8
Ferret – Scandisk Utility	55-10
Checktable Utility	55-12
Checktable – Levels of Checking	55-14
Checktable – Example	55-16
Table Rebuild Utility	55-18
Recovery Manager Utility	55-20
Recovery Manager Commands.....	55-22
Rcvmanager – List Status	55-24
Rcvmanager – List Locks	55-26
Rcvmanager – List Status (2 nd Example).....	55-28
Rcvmanager – List Rollback Tables.....	55-30
Rcvmanager – Cancel Rollback on Table	55-32
Vprocmanager.....	55-34
Showlocks Utility	55-36
Orphan or Phantom Spool Issues.....	55-38
Update Space Utility.....	55-40
Summary.....	55-42
Review Questions	55-44

Module 56 – Permanent Journals

Automatic Data Protection Mechanisms (Review).....	56-4
Permanent Journals	56-6
Location of Change Images	56-8
Assigning Tables to a Permanent Journal	56-10
Creating a Permanent Journal	56-12
Assigning a Permanent Journal	56-14
Before-Image Journals	56-16
After-Image Journals.....	56-18
Journal Subtables	56-20
Permanent Journal Statements	56-22
Recovery with Permanent Journals.....	56-24
Journals[x] View	56-26
Summary	56-28
Review Questions	56-30

Module 57 – A Tale of Three Tables

Permanent Journal Scenario.....	57-4
Table X.....	57-6
Table Y	57-8
Table Z	57-10
Permanent Journals	57-12
Archive Policy.....	57-14
Archive Scenario.....	57-16
After Restart Processing Completes.....	57-18
After REBUILD and Restart of Teradata.....	57-20
Table X Recovery	57-22
Table Y Recovery	57-24
Table Z Recovery	57-26
After Recovery	57-28
Summary	57-30

Module 58 – Archiving Data

Archive and Recovery Utility (ARC)	58-4
Archive and Recovery Phases	58-6
Restore versus FastLoad	58-8
ARC	58-10
Restart Log.....	58-12
Session Control.....	58-14
Multiple Sessions.....	58-16
ARC Statements.....	58-18
ARCHIVE Statement.....	58-20
ARCHIVE Examples.....	58-22
Archiving Selected Partitions of PPI Table	58-26
ARCHIVE Partition Example.....	58-28
ANALYZE Statement	58-30
ANALYZE Output	58-32
Archive Objects	58-34
Archive Objects (cont.).....	58-36
Archive Levels.....	58-38
Archive Levels (cont.)	58-40
Archive Options.....	58-42
Indexes Option.....	58-44
Group Read Lock Option.....	58-46
Database DBC Archive.....	58-48
Summary	58-50
Review Questions	58-52

Module 59 – Restoring Data

Understanding Restore Operations	59-4
Restore-Related Statements	59-6
The Restore Statement	59-8
Restoring Examples	59-10
RESTORE Example and Output.....	59-12
Restoring Selected Partitions of PPI Table	59-14
RESTORE Partition Example	59-16
COPY Statement	59-18
Copying Objects.....	59-20
Copying.....	59-22
BUILD Statement	59-24
RELEASE LOCK Statement	59-26
Revalidate References.....	59-28
Revalidate References Output.....	59-30
Recovery Control Data Dictionary Views	59-32
Association View	59-34
Events View	59-36
Restoring Data Summary	59-38
Review Questions	59-40

Module 60 – Data Recovery Operations

Data Recovery Using Roll Operations	60-4
The CHECKPOINT Statement	60-6
CHECKPOINT WITH SAVE Statement.....	60-8
Using the ROLLBACK Command	60-10
The ROLLBACK Statement	60-12
ROLLFORWARD Statement	60-14
ROLLFORWARD Restrictions	60-16
The ROLLFORWARD Statement	60-18
DELETE JOURNAL Statement	60-20
Summary	60-22
Review Questions	60-24

Module 61 – Teradata Factory Recap

Teradata Factory Review – Week 1.....	61-4
Teradata Factory Review – Week 2.....	61-6
Dictionary Tables to Maintain	61-8
Plan and Follow-up.....	61-10
Things You Never Have to do with Teradata.....	61-12
Things You Never Have to do with Teradata (cont.)	61-14
Teradata Differentiators (Part 1).....	61-16
Teradata Differentiators (Part 2).....	61-18
Teradata Certification – Materials to Focus on	61-20

Module 30



Additional Index Choices

After completing this module, you will be able to:

- List the three types of Join Indexes.
- Describe the situations where a Join Index can improve performance.
- Describe how to create a “sparse index”.
- Describe the difference between a single-table Join Index and a Hash Index.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Additional Index Choices.....	30-4
Join Indexes.....	30-6
Options for Join Indexes	30-8
Join Index Considerations	30-10
Join Index Example – Customer and Order Tables	30-12
Compressed Multi-Table Join Index.....	30-14
Non-Compressed Multi-Table Join Index.....	30-16
Example 1 – Does a Join Index Help?	30-18
Example 2 – Does a Join Index Help?	30-20
Join Index – Single Table.....	30-22
Join Index – Single Table (cont.)	30-24
Creating a Join Index – Single Table	30-26
Example 3 – Does the Join Index Help?	30-28
Why use Aggregate Join Indexes?	30-30
Aggregate Join Index Properties	30-32
Aggregation without an Aggregate Index	30-34
Creating an Aggregate Join Index.....	30-36
Aggregation with an Aggregate Index	30-38
Sparse Join Indexes	30-40
Creating a Sparse Join Index.....	30-42
Creating a Sparse Join Index on Partitions	30-44
Global (Join) Indexes	30-46
Global Index – Multiple Tables	30-48
Global Index as a “Hashed NUSI”.....	30-50
Creating a Global Index (“Hashed NUSI”)......	30-52
Example: Using a Global Index as a Hashed NUSI.....	30-54
Repeating Row Ids in Global Index	30-56
Hash Indexes	30-58
Creating a Hash Index	30-60
Hash Index – Example 2	30-62
Hash Index – Example 3	30-64
Hash Index – Example 3 (cont.)....	30-66
Summary	30-68
Review Questions	30-70
Lab Exercise 30-1	30-72

Additional Index Choices

As part of implementing a physical design, Teradata provides additional index choices that can improve performance for known queries and workloads. These will be described in more detail in this module.

Join indexes are defined to reduce the number of rows processed in generating result sets from certain types of queries, especially joins. Like secondary indexes, users may not directly access join indexes. They are an option available to the optimizer in query planning. The following are properties of join indexes:

- Are used to replicate and “pre-join” information from several tables into a single structure.
- Are designed to cover queries, reducing or eliminating the need for access to the base table rows.
- Usually do not contain pointers to base table rows (unless user defined to do so).
- Are distributed based on the user choice of a Primary Index on the Join Index.
- Permit Secondary Indexes to be defined on the Join Index (except for Single Table Join Indexes), with either “hash” or “value” ordering.

Unlike traditional indexes, join indexes do not store “pointers” to their associated base table rows. Instead, they are a fast path *final* access point that eliminates the need to access and join the base tables they represent. They substitute *for* rather than point *to* base table rows.

The Optimizer can choose to resolve a query using the index, rather than performing a join of two or more tables. Depending on the complexity of the join, this improves query performance considerably. To improve the performance of Join Index creation and Join Index maintenance during updates, consider collecting statistics on the base tables of a Join Index.



Additional Index Choices

As part of the physical design process, the designer may choose to implement join and/or hash indexes for performance reasons.

- **Join Indexes**

- Can be created to pre-join multiple tables.
- Can be used on a single table to redistribute the table on a different column – effectively creating an alternative primary index on a foreign key column.
- Can be used as a summary table to aggregate one or more columns from a table or tables.

- **Hash Indexes**

- Contains properties of both secondary indexes and single table join indexes.

- **Both Join Indexes and Hash Indexes**

- Provides the optimizer with additional options and the optimizer may use the join index if it “covers” the query.
- For known queries, this typically will result in much better performance.

Join Indexes

There are multiple ways in which a join index can be used. Three common ways include:

- Single table Join Index — Distribute the rows of a single table on the hash value of a foreign key value
- Multi-Table Join Index — Pre-join multiple tables; stores and maintains the result from joining two or more tables.
- Aggregate Join Index — Aggregate one or more columns of a single table or multiple tables into a summary table

A join index is a system-maintained index table that stores and maintains the joined rows of two or more tables (**multiple table join index**) and, optionally, aggregates selected columns, referred to as an **aggregate join index**.

Join indexes are defined in a way that allows join queries to be resolved without accessing or joining their underlying base tables. A join index is useful for queries where the index structure contains all the columns referenced by one or more joins, thereby allowing the index to cover all or part of the query. For obvious reasons, such an index is often referred to as a covering index. Join indexes are also useful for queries that aggregate columns from tables with large cardinalities. These indexes play the role of pre-join and summary tables without denormalizing the logical design of the database and without incurring the update anomalies presented by denormalized tables.

Another form of join index, referred to as a **single table join index**, is very useful for resolving joins on large tables without having to redistribute the joined rows across the AMPs.



Join Indexes

- A Join Index is an optional index which may be created by the user. The basic types of Join Indexes will be described first.
- **Multi-table Join Index**
 - Pre-join multiple tables; stores and maintains the result from joining two or more tables.
 - Facilitates join operations by possibly **eliminating join processing** or by **reducing/eliminating join data redistribution**.
- **Single-table Join Index**
 - Distribute the rows of a single table on the hash value of a foreign key value.
 - Facilitates the ability to join the foreign key table with the primary key table **without redistributing the data**.
- **Aggregate Join Index (AJI)**
 - Aggregate (SUM or COUNT) one or more columns of a single table or multiple tables into a summary table.
 - Facilitates aggregation queries by **eliminating aggregation processing**. The pre-aggregated values are contained in the AJI instead of relying on base table calculations.

Options for Join Indexes

The facing page highlights two options that are available with join indexes.

A Sparse Join Index is simply a term that is used when a join index is created with a WHERE condition. You can use a WHERE clause in the CREATE JOIN INDEX statement to limit the rows that are created in the join index. This effectively reduces the size (PERM space) of the join index. The rows included in the join index are a subset of the rows in the base table or tables based on an SQL query result.

A Global Index is simply a term that is used to define a join index that contains the Row IDs of the base table rows. This means that the user includes the ROWID as a user-defined column within the join index. Row IDs that are included within a join index are always 10 bytes in length, regardless if the base table is partitioned or not.



Options for Join Indexes

- **Sparse Join Indexes**

- When creating any of the join indexes, you can include a WHERE clause to limit the rows created in the join index to a subset of the rows in the base table or tables.

- **Global Join Indexes**

- You can include the Row ID of the table(s) within the join index to allow an AMP to join back to the data row for columns not referenced (covered) in the join index.

- **Miscellaneous notes:**

- Materialized Views are implemented as Join Indexes. Join Indexes improve query performance at the expense of slower updates and increased storage.
 - When you create a Join Index, there is no duplicate row check – you don't have to worry about a high number of NUPI duplicates in the join index.

Join Index Considerations

Join Index considerations include:

- You cannot specify a column with a data type of either BLOB or CLOB in the definition of a join index (nor for any other kind of index).
- Be aware that when you define a join index using an outer join, you must reference all the columns of the outer table in the select list of the join index definition. If any of the outer table columns are not referenced in the select list for the join index definition, the system returns an error to the requestor.
- Perm Space — a Join Index is created as a table-like structure in Teradata and uses Perm space.
- Fallback Protection — Join Index subtables can be Fallback-protected starting with Teradata V2R4 software.
- Load Utilities — MultiLoad and FastLoad utilities cannot be used to load or unload data into base tables that have an associated join index defined on them because join indexes are not maintained during the execution of these utilities. If an error occurs, the join index must be dropped and recreated after that table has been loaded. The TPump utility, which performs standard SQL row inserts and updates, can be used because join indexes are properly maintained during the execution of such utilities.
- Archive and Restore — archiving is permitted on a base table or database that has a Join Index. During a restore of such a base table or database, the system does not automatically rebuild the Join Index. Instead, the Join Index is marked as invalid.
- Permanent Journal Recovery — using a permanent journal to recover a base table (i.e., ROLLBACK or ROLLFORWARD) with an associated Join Index defined is permitted. The join index is not automatically rebuilt during the recovery process. Instead, the join index is marked as invalid and the join index must be dropped and recreated before it can be used again in the execution of queries.
- Collecting Statistics — statistics should be collected on the primary index and secondary indexes of the Join Index to provide the Optimizer with baseline statistics, including the total number of rows in the Join Index.



Join Index Considerations

Join Index considerations include:

- Join indexes are updated automatically as base tables are updated (additional I/O).
- Take up PERM space and may (or may not) be Fallback protected.
- You can specify no more than 64 columns per referenced base table per join index.
- BLOB and CLOB data types **cannot** be defined within a Join Index.
- A Trigger and a Join Index **cannot** be defined on the same table on releases before Teradata V2R6.2.
- Load utilities such as MultiLoad and FastLoad can't be used (use TPump).
- Archive and Restore – after restoring tables, drop and recreate the Join Index.
- Permanent Journals Recovery of the Join Index is **not** supported.

In many respects, a Join Index is similar to a base table.

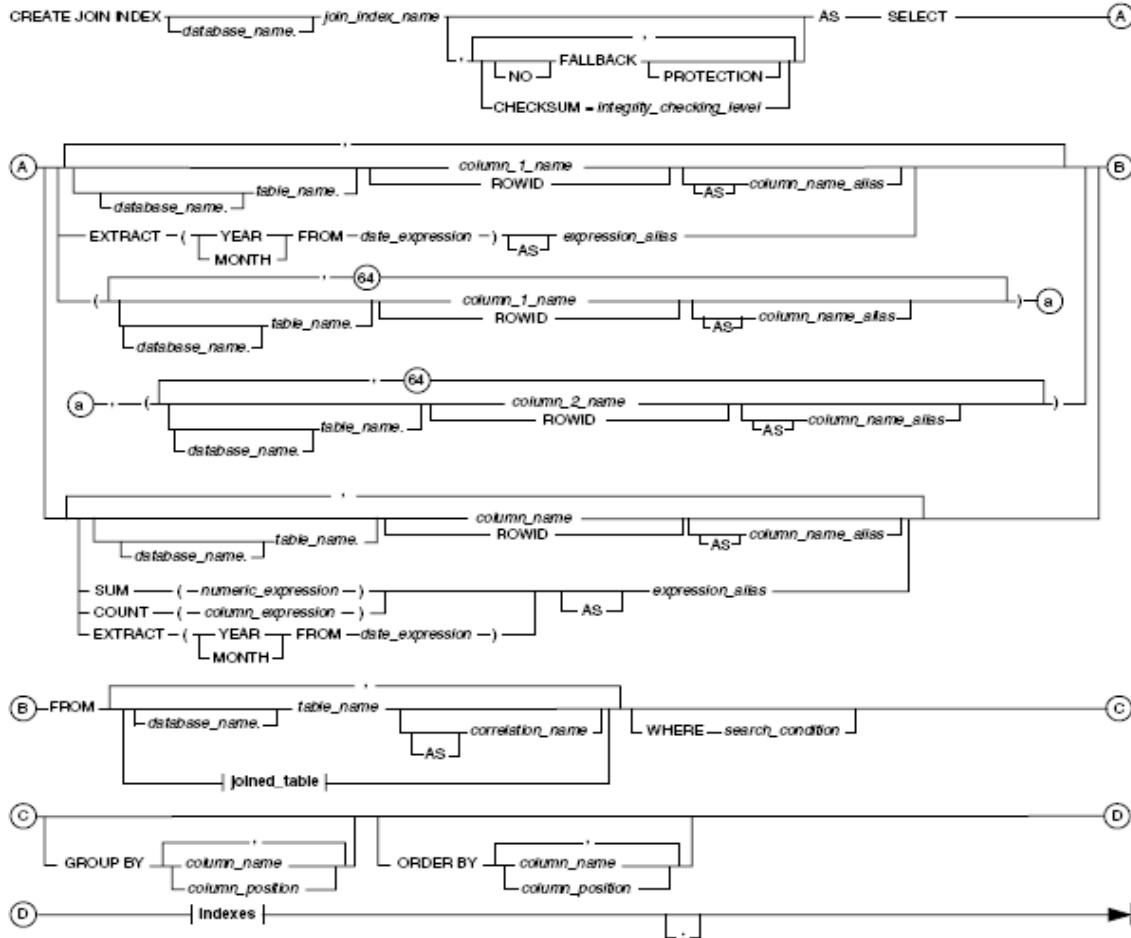
- You may create non-unique secondary indexes on its columns.
- Perform COLLECT/DROP STATISTICS, DROP, HELP, and SHOW statements.

Unlike base tables, you **cannot** do the following:

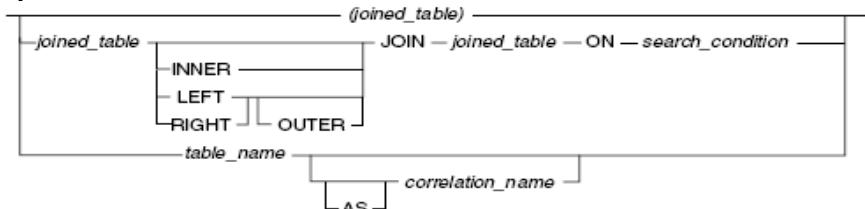
- Directly query or update join index rows.
- Create unique indexes on its columns.
- Store and maintain arbitrary query results such as expressions.

Join Index Example – Customer and Order Tables

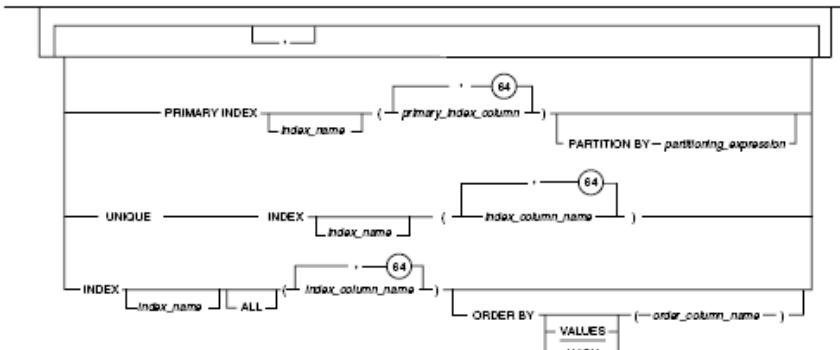
The CREATE JOIN INDEX syntax is shown below.



Joined_Table Excerpt



Indexes Excerpt

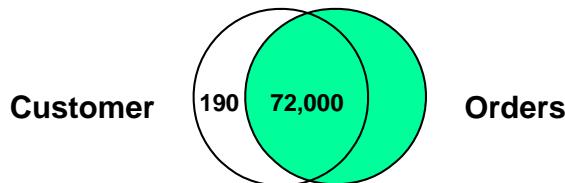




Join Index Example Customer and Order Tables

```
CREATE SET TABLE Customer
( c_custid      INTEGER NOT NULL,
  c_lname       VARCHAR(15),
  c_fname       VARCHAR(10),
  c_address     VARCHAR(50),
  c_city        VARCHAR(20),
  c_state       CHAR(2),
  c_zipcode     INTEGER)
UNIQUE PRIMARY INDEX ( c_custid );
```

```
CREATE SET TABLE Orders
( o_orderid     INTEGER NOT NULL,
  o_custid      INTEGER NOT NULL,
  o_orderstatus  CHAR(1),
  o_totalprice   DECIMAL(9,2) NOT NULL,
  o_orderdate    DATE
                FORMAT 'YYYY-MM-DD' NOT NULL,
  o_orderpriority SMALLINT,
  o_clerk        CHAR(16),
  o_shippriority SMALLINT,
  o_comment      VARCHAR(79))
UNIQUE PRIMARY INDEX ( o_orderid );
```



**5000 Customers and 72,000 Orders.
72,000 orders are associated with valid customers.
190 customers have no orders.**

Compressed Multi-Table Join Index

The facing page includes an example of creating a Multiple Table Join Index using the repeating group option.

The storage organization for join indexes supports a compressed format to reduce storage space.

If you know that a join index contains groups of rows with repeating information, then its definition DDL can specify repeating groups, indicating the repeating columns in parentheses. The column list is specified as two groups of columns, with each group stipulated within parentheses. The first group contains the fixed columns and the second group contains the repeating columns.

As another option, you can elect to store join indexes in value order, ordered by the values of a 4-byte column. Value-ordered storage provides better performance for queries that specify selection constraints on the value ordering column. For example, suppose a common task is to look up sales information by order date. You can create a join index on the Orders table and order it by order date. The benefit is that queries that request orders by order date only need to access those data blocks that contain the value or range of values that the queries specify.

Physical Join Index Row and Compression

A physical join index row has two parts:

- A required fixed part that is stored only once.
- An optional repeated part that is stored as often as needed.

For example, if a logical join result has n rows with the same fixed part value, then there is one corresponding physical join index row that includes n repeated parts in the physical join index. A physical join index row represents one or more logical join index rows. The number of logical join index rows represented in the physical row depends on how many repeated values are stored.

Limitations

For a compressed multi-table join index, the maximum number of columns defined in the fixed portion is 64 and the maximum number of columns defined in the repeating portion is also 64. The total maximum number of columns in this type of join index is 128.

With a LEFT or RIGHT OUTER join, at least 1 column from each inner table must be NOT NULL.

FULL OUTER joins are not allowed with a compressed multi-table join index.



Compressed Multi-Table Join Index

```
CREATE JOIN INDEX
  SELECT
    FROM
      INNER JOIN
        ON
          PRIMARY INDEX
```

Cust_Ord_JI AS
`(c_custid, c_lname),`
`(o_orderid, o_orderstatus, o_orderdate)`
Customer C
Orders O
`c_custid = o_custid`

Fixed portion of Join Index
(max # columns is 64)

Repeating portion of Join Index
(max # of columns is 64)

Maximum # of columns for
compressed join index is 128.

Fixed Portion

Variable Portion

c_custid	c_lname	o_orderid	o_orderstatus	o_orderdate
1443	Woods	102292	C	2002-11-30
		135893	C	2005-11-30
		157093	O	2006-09-16
		135993	C	2005-12-14
4000	Mickelson	142000	C	2005-12-31
		129600	C	2003-12-29
		154798	C	2006-04-17
		122398	C	2003-12-31
5809	Furyk	133599	C	2004-12-31
		101199	C	2002-01-11
		154999	O	2006-08-24
		122399	C	2003-06-31

Within the join index, this
is one row of data
representing the orders for
a customer.

One row in Join Index.

One row in Join Index.

Non-Compressed Multi-Table Join Index

The facing page includes an example of creating a Multiple Table Join Index without using the repeating group option.

The storage space in this example for the non-compressed multi-table join index will be higher.

For a non-compressed multi-table join index, the maximum number of columns defined per referenced table is 64. The total maximum number of columns in this type of join index is 2048.

PERM Space Required

The amount of PERM space used by the compressed multi-table join index (previous example) and the non-compressed join index is listed below. Remember that these tables are quite small and note that the join index with repeating data requires less storage space.

<u>TableName</u>	<u>SUM(CurrentPerm)</u>
Cust_Ord_JI	1,044,480
Cust_Ord_JI2	2,706,432



Non-Compressed Multi-Table Join Index

```
CREATE JOIN INDEX Cust_Ord_JI2 AS
  SELECT c_custid, c_lname,
         o_orderid, o_orderstatus, o_orderdate
    FROM Customer C
   INNER JOIN Orders O
      ON c_custid = o_custid
 PRIMARY INDEX (c_custid);
```

Max # columns per referenced table is 64

Maximum # of columns for non-compressed join index is 2048.

Fixed Portion

Variable Portion

c_custid	c_lname	o_orderid	o_orderstatus	o_orderdate
1443	Woods	102292	C	2002-11-30
1443	Woods	135893	C	2005-11-30
1443	Woods	157093	O	2006-09-16
1443	Woods	135993	C	2005-12-14
4000	Mickelson	142000	C	2005-12-31
4000	Mickelson	129600	C	2003-12-29
4000	Mickelson	154798	C	2006-04-17
4000	Mickelson	122398	C	2003-12-31
5809	Furyk	133599	C	2004-12-31
5809	Furyk	101199	C	2002-01-11
5809	Furyk	154999	O	2006-08-24
5809	Furyk	122399	C	2003-06-31

Within the join index, each order is represented by a separate join index row.

PERM Space Used

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Cust_Ord_JI2	2,706,432

Example 1 – Does a Join Index Help?

This EXPLAIN is without a Join Index.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.O for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("TFACT.O.o_orderstatus = 'O'") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 3,970 rows. The estimated time for this step is 0.41 seconds.
 - 5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.c_custid = o_custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.07 seconds.
 - 6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.49 seconds.

This EXPLAIN is with a Join Index.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.CUST_ORD_JI.
 - 2) Next, we lock TFACT.CUST_ORD_JI for read.
 - 3) We do an all-AMPs RETRIEVE step from TFACT.CUST_ORD_JI by way of an all-rows scan with a condition of ("TFACT.CUST_ORD_JI.o_orderstatus = 'O'") into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with no confidence to be 3970 rows. The estimated time for this step is 0.18 seconds.
 - 4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.18 seconds.



Example 1 – Does a Join Index Help?

List the valid customers who have open orders?

```
SELECT      c_custid, c_Iname, o_orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          c_custid = o_custid
WHERE       o_orderstatus = 'O'
ORDER BY    1;
```

<u>SQL Query</u>	<u>Time</u>
Without Join Index	.49 seconds
With Join Index	.18 seconds

c_custid	c_Iname	o_orderdate
1386	Poppy	2006-10-26
1906	Putman	2006-10-31
1969	Mitchell	2006-12-07
2916	Rotter	2006-11-22
2954	Agnew	2006-12-02
4336	Carson	2006-09-12
5396	Murphy	2006-10-29
:	:	:

All referenced columns are part of the Join Index.
 Optimizer picks Join Index rather than doing a join.
 Join Index covers query and helps this query.
 The Compressed Join Index was used for this example.

Explains are provided on facing page in the PDF file.

Example 2 – Does a Join Index Help?

This EXPLAIN is without a Join Index.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.O for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("TFACT.O.o_orderstatus = 'O'") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 3,970 rows. The estimated time for this step is 0.41 seconds.
 - 5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.c_custid = o_custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.07 seconds.
 - 6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.49 seconds.

This EXPLAIN is with a Join Index.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.CUST_ORD_JI.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.CUST_ORD_JI for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs JOIN step from TFACT.C by way of a RowHash match scan with no residual conditions, which is joined to TFACT.CUST_ORD_JI by way of a RowHash match scan with a condition of ("TFACT.CUST_ORD_JI.o_orderstatus = 'O'"). TFACT.C and TFACT.CUST_ORD_JI are joined using a merge join. The input table TFACT.CUST_ORD_JI will not be cached in memory. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.25 seconds.
 - 5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.25 seconds.



Example 2 – Does a Join Index Help?

List the valid customers and their addresses who have open orders?

```
SELECT      c_custid, c_Iname,
            c_address, c_city, c_state,
            o_orderdate
        FROM Customer C
    INNER JOIN Orders O
        ON c_custid = o_custid
       WHERE o_orderstatus = 'O'
      ORDER BY 1;
```

SQL Query

Without Join Index

Time

.49 seconds

With Join Index

.25 seconds

- Some of the referenced columns are NOT part of the Join Index. The Join Index does not cover the query, but is used in this example.
- A Join Index is used in this query and is merge joined with the Customer table.

Results:

c_custid	c_Iname	c_address	c_city	c_state	o_orderdate
1391	Poppy	2300 Madrona Ave	Carson City	NV	2006-10-27
1906	Putman	903 La Pierre Ave	Jackson	MS	2006-10-31
1969	Mitchell	36 Main Street	New York	NY	2006-12-07
2916	Rotter	4564 Long Beach Blvd	Trenton	NJ	2006-11-22
2954	Agnew	1083 Beryl Ave	Los Angeles	CA	2006-12-02
4336	Carson	4021 Eleana Way	St. Paul	MN	2006-09-12
5396	Murphy	5603 Main Street	Pierre	SD	2006-10-29
:	:	:	:	:	:

Explains are provided on facing page in the PDF file.

Join Index – Single Table

A denormalization technique is to replicate a column in a table to avoid joins. If an SQL query would benefit from replicating some or all of its columns in another table that is hashed on the join field (usually the primary index of the table to which it is to be joined) rather than the primary index of the original base table, then you should consider creating one or more single table join indexes on that table.

For example, you might want to create a single table join index to avoid redistributing a large base table or to avoid the possibly prohibitive storage requirements of a multi-table join index. For example, a single table join index might be useful for commonly made joins having low predicate selectivity but high join selectivity.

Join Index – Single Table

- The **Single Table Join Index** is useful for resolving joins on large tables without having to redistribute the joined rows across the AMPs.
- In some cases, this may perform better than building a multi-table join index on the same columns.

Orders	
<u>o_orderid</u>	<u>o_custid</u>
PK	FK
UPI	

Without a Join Index, redistribution or duplication is required to complete the join of Orders and Customer (or Customer History).

Order_Line_Item	
<u>o_orderid</u>	<u>o_part_id</u>
PK	
NUPI	

Shipments	
<u>s_shipid</u>	<u>s_orderid</u>
PK	FK
	UPI

Customer	
<u>c_custid</u>	
PK	
UPI	

Customer_History	
<u>c_custid</u>	
PK	
UPI	

Join Index – Single Table (cont.)

You can also define a simple join index on a single table. This permits you to hash some or all of the columns of a large replicated base table on a foreign key that hashes rows to the same AMP as another large table. In some situations, this may perform better than building a multi-table join index on the same columns. The advantage comes from less under-the-covers update maintenance on the single table form of the index. Only testing can determine which is the better design for a given set of tables, applications, and hardware configuration.

The example on the facing page shows a technique where the join index is effectively substituted for the underlying base table. The join index has a primary index that ensures that rows are hashed to the same AMPs as rows in tables being joined. This eliminates the need for row redistribution when the join is made.

Even though each single table join index you create partly or entirely replicates its base table, you cannot query or update them directly just as you cannot directly query or update any other join index.

In this example, the compressed format for a single table join index can be used.

```
CREATE JOIN INDEX Orders_JI2 AS
SELECT (o_custid),
          (o_orderid,
           o_orderstatus,
           o_totalprice,
           o_orderdate)
FROM Orders
PRIMARY INDEX (o_custid);
```

PERM Space Required

The amount of PERM space used by the compressed multi-table join index (previous example) and the single table join index is listed below. Remember that these tables are quite small and note that the join index with repeating data requires less storage space.

<u>TableName</u>	<u>SUM(CurrentPerm)</u>
Cust_Ord_JI	1,044,480
Orders_JI	1,238,584
Orders_JI2	2,308,608

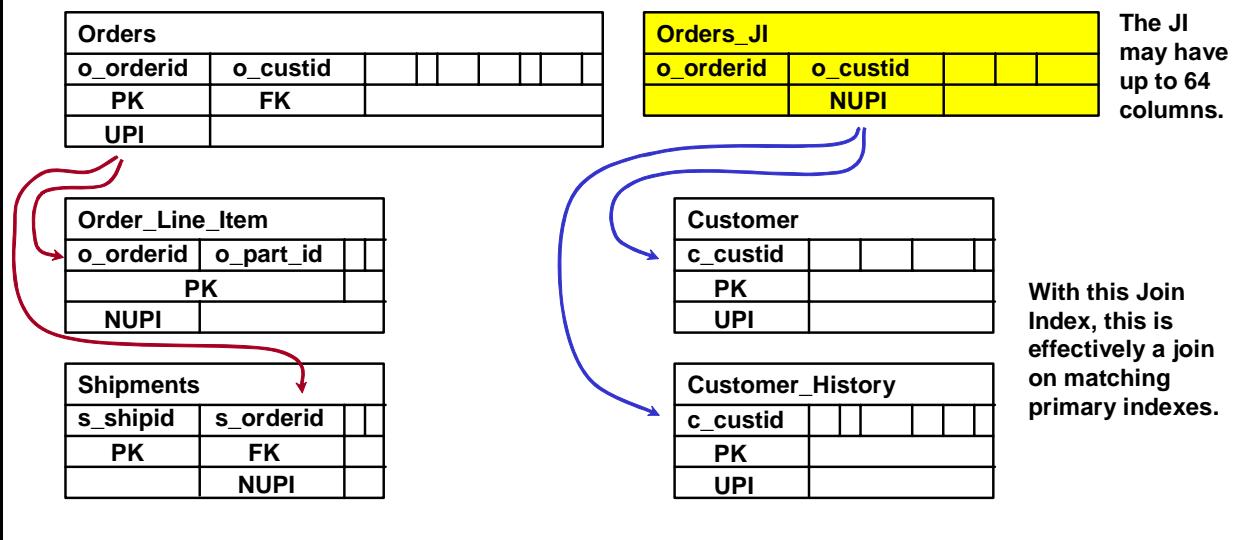
Note that in this example that the single table join index uses more permanent space. However, the single table join index has some columns (from the Orders table) that not part of the compressed multi-table join index.



Join Index – Single Table (cont.)

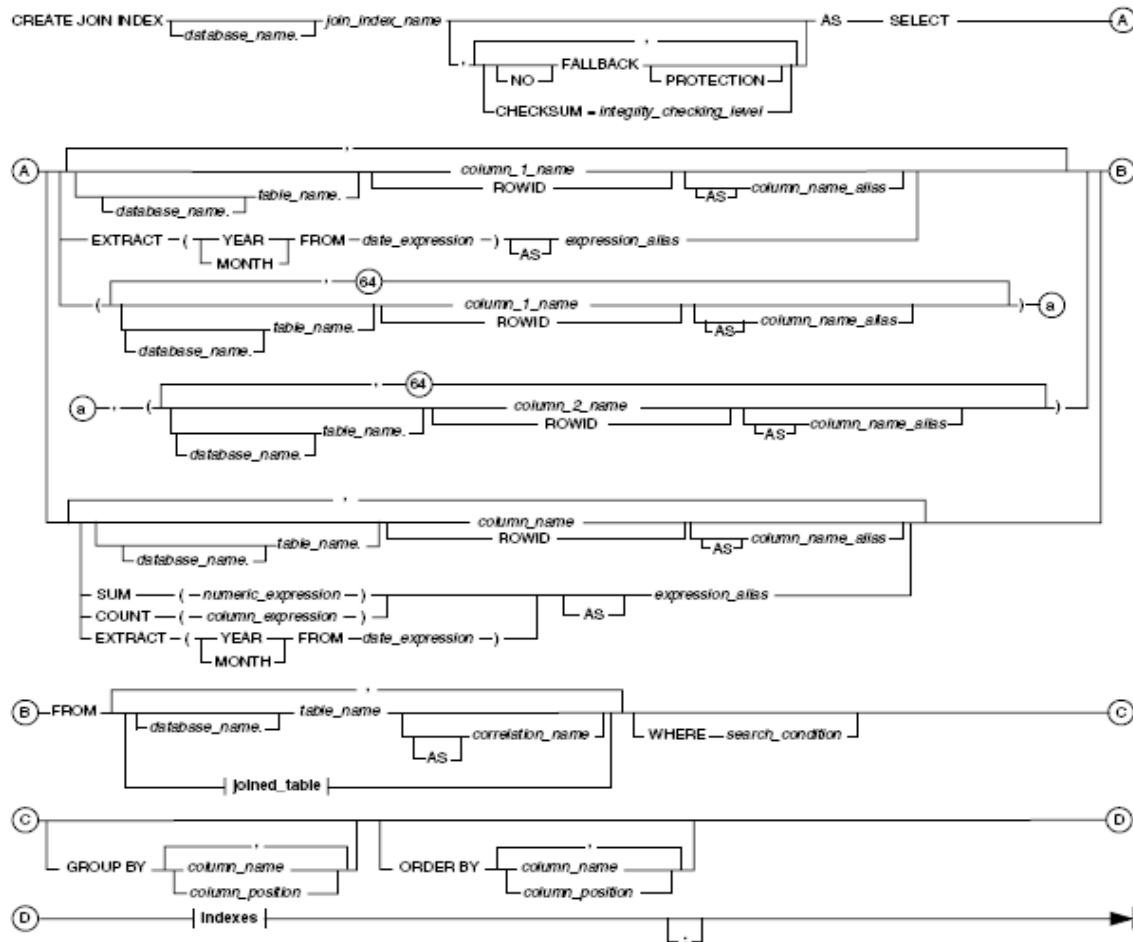
- Possible advantages include:

- Less update maintenance on the single table Join Index than a multi-table Join Index.
- Maybe less storage space for a single table Join Index than for a multi-table Join Index.

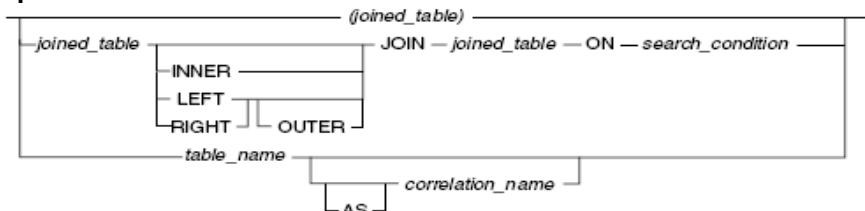


Creating a Join Index – Single Table

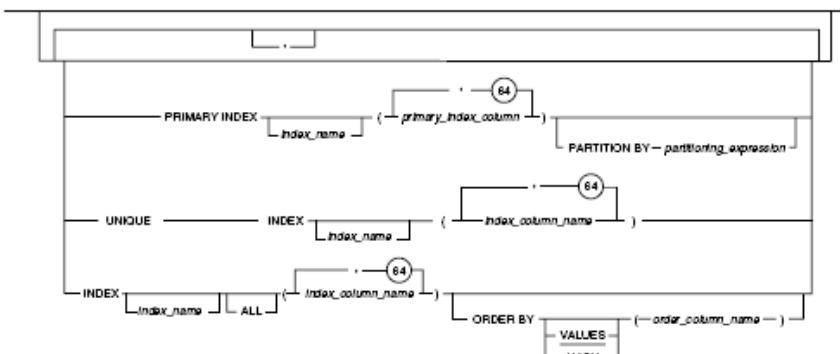
The CREATE JOIN INDEX syntax is shown below.



Joined_Table Excerpt



Indexes Excerpt





Creating a Join Index – Single Table

Compressed

```
CREATE JOIN INDEX Orders_JI AS
SELECT          (o_custid),
                (o_orderid,
                 o_orderstatus,
                 o_totalprice,
                 o_orderdate)
FROM            Orders
PRIMARY INDEX  (o_custid);
```

Non-Compressed

```
CREATE JOIN INDEX Orders_JI2 AS
SELECT          o_orderid,
                o_custid,
                o_orderstatus,
                o_totalprice,
                o_orderdate
FROM            Orders
PRIMARY INDEX  (o_custid);
```

The **Orders** base table is distributed across the AMPs based on the hash value of the ***o_orderid*** column (primary index of base table).

The Join Index (**Orders_JI**) effectively represents a subset of the **Orders** table (selected columns) and is distributed across the AMPs based on the hash value of the ***o_custid*** column.

The optimizer can use this Join Index to improve joins using the “**customer id**” to join with the **Orders** table.

Example 3 – Does the Join Index Help?

This EXPLAIN is without a Join Index.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.O for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("TFACT.O.o_orderstatus = 'O'") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with high confidence to be 3,970 rows. The estimated time for this step is 0.41 seconds.
 - 5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.c_custid = o_custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.07 seconds.
 - 6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.49 seconds.

This EXPLAIN is with a Single Table Join Index (compressed single table join index).

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.ORDERS_JI.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.ORDERS_JI for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs JOIN step from TFACT.C by way of a RowHash match scan with no residual conditions, which is joined to TFACT.ORDERS_JI by way of a RowHash match scan with a condition of ("TFACT.ORDERS_JI.o_orderstatus = 'O'"). TFACT.C and TFACT.ORDERS_JI are joined using a merge join, with a join condition of ("TFACT.C.c_custid = TFACT.ORDERS_JI.o_custid"). The input table TFACT.ORDERS_JI will not be cached in memory. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 3,970 rows. The estimated time for this step is 0.22 seconds.
 - 5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.22 seconds.

Note: The EXPLAIN cost estimates were the same for both Orders_JI (compressed join index) and Orders_JI2 (non-compressed join index).



Example 3 – Does the Join Index Help?

List the valid customers who have open orders?

```
SELECT      c_custid, c_Iname, o_orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          c_custid = o_custid
WHERE       o_orderstatus = 'O'
ORDER BY    1;
```

<u>SQL Query</u>	<u>Time</u>
Without Join Index	.49 seconds
With Join Index	.22 seconds

c_custid	c_Iname	o_orderdate
1386	Poppy	2006-10-26
1906	Putman	2006-10-31
1969	Mitchell	2006-12-07
2916	Rotter	2006-11-22
2954	Agnew	2006-12-02
4336	Carson	2006-09-12
5396	Murphy	2006-10-29
:	:	:

- The rows of the customer table and the Join Index are located on the same AMP.
- A single table Join Index will help this query.

Explains are provided on facing page in the PDF file.

Why use Aggregate Join Indexes?

Summary Tables

Queries that involve counts, sums, or averages over large tables require processing to perform the needed aggregations. If the tables are large, query performance may be affected by the cost of performing the aggregations. Traditionally, when these queries are run frequently, users have built summary tables to expedite their performance. While summary tables do help query performance there are disadvantages associated with them as well.

Summary Tables Limitations

- Require the creation of a separate table
- Require initial population of the table
- Require refresh of changing data, either via update or reload
- Require queries to be coded to access summary tables, not the base tables
- Allow for multiple versions of the truth when the summary tables are not up-to-date

Aggregate Indexes

The primary function of an aggregate join index is to provide the Optimizer with a performance, cost-effective means for satisfying any query that specifies a frequently made aggregation operation on one or more columns. The aggregate join index permits you to define a summary table without violating schema normalization.

Aggregate indexes provide a solution that enhances the performance of the query while reducing the requirements placed on the user. All of the above listed limitations are overcome with their use.

An aggregate index is created similarly to a join index with the difference that sums, counts and date extracts may be used in the definition. A denormalized summary table is internally created and populated as a result of creation. The index can never be accessed directly by the user. It is available only to the optimizer as a tool in its query planning.

Aggregate indexes do not require any user maintenance. When underlying base table data is updated, the aggregate index totals are adjusted to reflect the changes. While this requires additional processing overhead when a base table is changed, it guarantees that the user will have up-to-date information in the index.



Why use Aggregate Join Indexes?

Summary Tables

- Queries involving aggregations over large tables are subject to high compute and I/O overhead. Summary tables often used to expedite their performance.

Summary Tables Limitations

- Require the creation of a separate summary table.
- Require initial population of the summary table.
- Requires refresh of summary table.
- Queries must access summary table, not the base table.
- Multiple “versions of the truth”.

Aggregate Join Indexes

- Aggregate join indexes enhance the performance of the query while reducing the requirements placed on the user.
- An aggregate join index is created similarly to a join index with the difference that sums, counts and date extracts may be used in the definition.

Aggregate Join Index Advantages

- Do not require any user maintenance.
- Updated automatically when base tables change (requires processing overhead)
- User will have up-to-date information in the index.

Aggregate Join Index Properties

Aggregate Indexes are similar to other Join Indexes in that they are:

- Automatically kept up to date without user involvement.
- Never accessed directly by the user.
- Optional and provide an additional choice for the optimizer.
- MultiLoad and FastLoad may not be used to load tables for which indexes are defined.

Aggregate Indexes are different from other Join Indexes in that they:

- Use the SUM and COUNT functions.
- Permit use of EXTRACT YEAR and EXTRACT MONTH from dates.

Define an aggregate join index as a join index that specifies SUM or COUNT aggregate operations. No other aggregate functions are permitted in the definition of a join index.

To avoid numeric overflow, the COUNT and SUM fields in a join index definition must be typed as FLOAT. If you do not assign a data type to COUNT and SUM, the system types them as FLOAT automatically. If you assign a type other than FLOAT, an error message occurs.

You must have one of the following two privileges to create any join index:

- CREATE TABLE on the database or user which will own the join index,
or
- INDEX privilege on each of the base tables.

Additionally, you must have this privilege:

- DROP TABLE rights on each of the base tables.

The following table will be used in the subsequent examples:



Aggregate Join Index Properties

Aggregate Indexes are similar to other Join Indexes:

- Automatically kept up to date without user involvement.
- Never accessed directly by the user.
- Optional and provide an additional choice for the optimizer.
- MultiLoad and FastLoad may NOT be used to load tables for which indexes are defined.

Aggregate Indexes differ from other Join Indexes:

- Use the SUM and COUNT functions.
- Permit use of EXTRACT YEAR and EXTRACT MONTH from dates.

Privileges required to create any Join Index:

- CREATE TABLE in the database or user which will own the join index, or INDEX privilege on each of the base tables.

Additionally, you must have this privilege:

- DROP TABLE rights on each of the base tables.

Aggregation without an Aggregate Index

The facing page contains an example of aggregation and the base table does NOT have an aggregate index.

The Daily_Sales table has 35 item_ids and a row for every item for every day from 2002 through 2007.

The Daily_Sales table has 76,685 rows (2191 days x 35 items).

Note: Statistics were collected for all of the columns on the Daily_Sales table.

```
EXPLAIN SELECT
    item_id
    ,EXTRACT (YEAR FROM sales_date) AS Yr
    ,EXTRACT (MONTH FROM sales_date) AS Mon
    ,SUM (sales)
FROM Daily_Sales
GROUP BY 1, 2, 3
ORDER BY 1, 2, 3;
```

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.Daily_Sales.
 - 2) Next, we lock TFACT.Daily_Sales for read.
 - 3) We do an all-AMPs SUM step to aggregate from TFACT.Daily_Sales by way of an all-rows scan with no residual conditions, and the grouping identifier in field 1. Aggregate Intermediate Results are computed locally, then placed in Spool 3. The input table will not be cached in memory, but it is eligible for synchronized scanning. The aggregate spool file will not be cached in memory. The size of Spool 3 is estimated with low confidence to be 57,514 rows. The estimated time for this step is 0.85 seconds.
 - 4) We do an all-AMPs RETRIEVE step from Spool 3 (Last Use) by way of an all-rows scan into Spool 1 (group_amps), which is built locally on the AMPS. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 57,514 rows. The estimated time for this step is 0.39 seconds.
 - 5) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1.



Aggregation without an Aggregate Index

List the sales by Year and Month for every item.

```

SELECT      item_id
            ,EXTRACT (YEAR FROM sales_date) AS Yr
            ,EXTRACT (MONTH FROM sales_date) AS Mon
            ,SUM (sales)
FROM        Daily_Sales
GROUP BY    1, 2, 3
ORDER BY    1, 2, 3;

```

An all-rows scan of base table is required.
The base table has 76,685 rows.

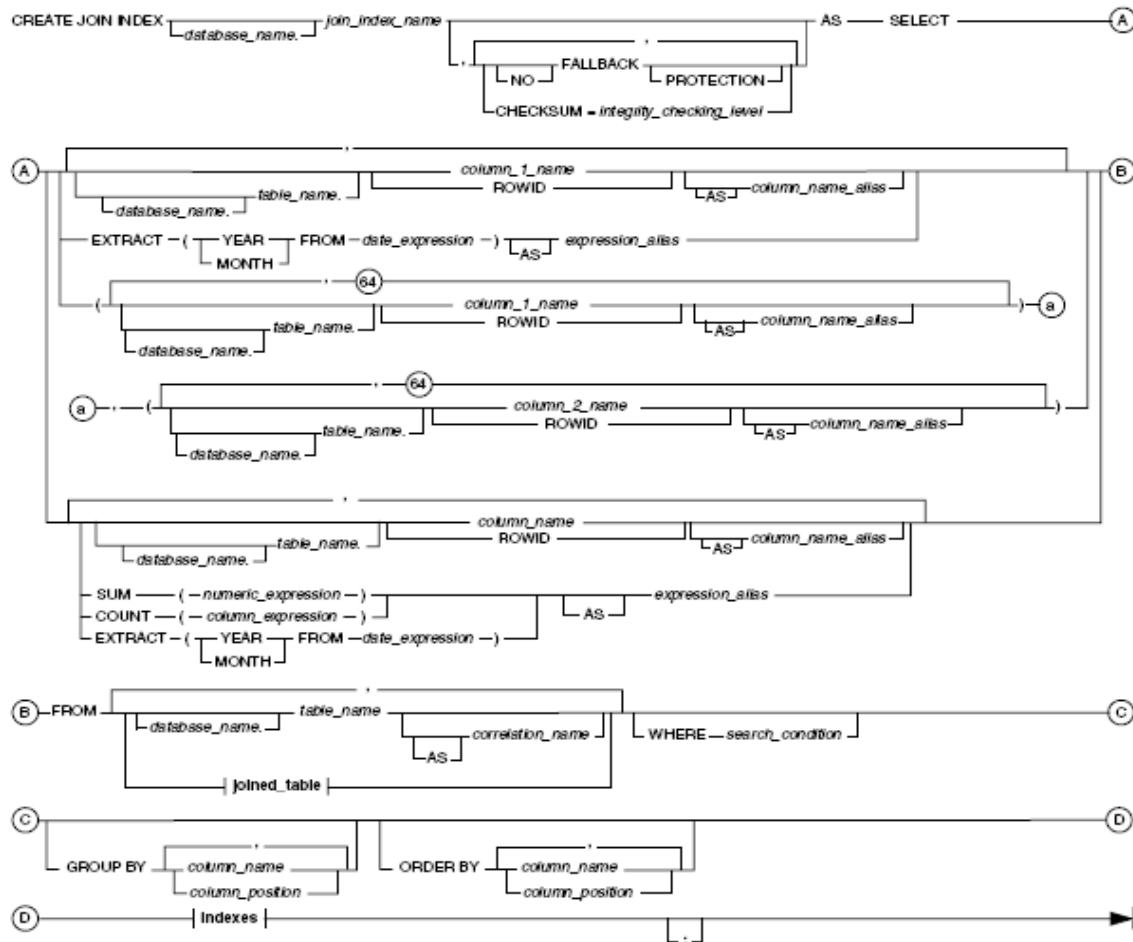
EXPLAIN without an Aggregate Index (Partial Listing)

V2R6.2 EXPLAIN

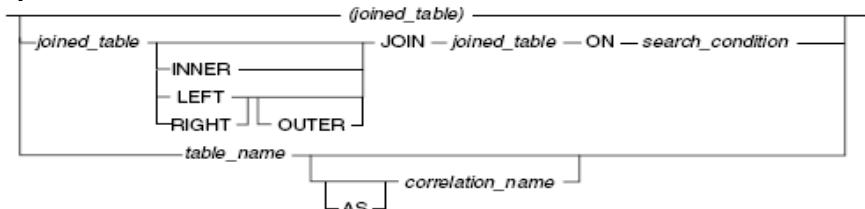
- :
 3) We do an all-AMPs SUM step to aggregate from TFACT.Daily_Sales by way of an all-rows scan with no residual conditions, and the grouping identifier in field 1. Aggregate Intermediate Results are computed locally, then placed in Spool 3. The input table will not be cached in memory, but it is eligible for synchronized scanning. The aggregate spool file will not be cached in memory. The size of Spool 3 is estimated with low confidence to be 57,514 rows. **The estimated time for this step is 0.85 seconds.**
 4) We do an all-AMPs RETRIEVE step from Spool 3 (Last Use) by way of an all-rows scan into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 57,514 rows. **The estimated time for this step is 0.39 seconds.**
 :
 :

Creating an Aggregate Join Index

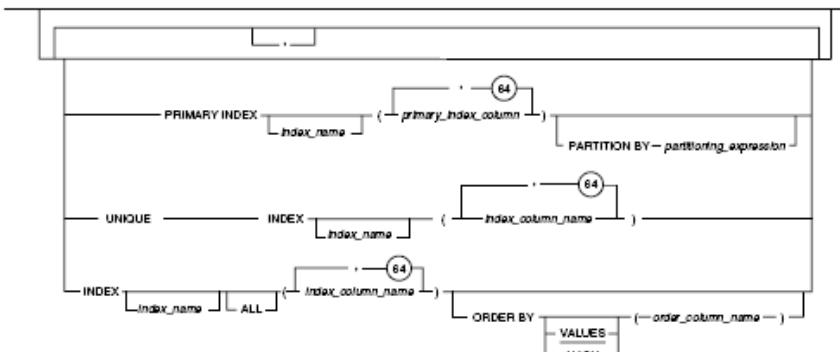
The CREATE JOIN INDEX syntax is shown below.



Joined_Table Excerpt



Indexes Excerpt





Creating an Aggregate Join Index

```
CREATE TABLE Daily_Sales
( item_id      INTEGER NOT NULL
,sales_date    DATE FORMAT 'yyyy-mm-dd'
,sales        DECIMAL(9,2) )
PRIMARY INDEX (item_id);
```

```
CREATE JOIN INDEX Monthly_Sales_JI AS
SELECT
    item_id AS Item
    ,EXTRACT (YEAR FROM sales_date) AS Yr
    ,EXTRACT (MONTH FROM sales_date) AS Mon
    ,SUM (sales) AS Sum_of_Sales
FROM Daily_Sales
GROUP BY 1, 2, 3;
```

```
COLLECT STATISTICS ON Monthly_Sales_JI COLUMN Item;
COLLECT STATISTICS ON Monthly_Sales_JI COLUMN Yr ;
COLLECT STATISTICS ON Monthly_Sales_JI COLUMN Mon;

HELP STATISTICS Monthly_Sales_JI;
```

Date	Time	Unique Values	Column Names
08/01/21	18:43:13	35	Item
08/01/21	18:43:13	6	Yr
08/01/21	18:43:13	12	Mon

Aggregation with an Aggregate Index

Execution of the following SELECT yields the result below:

```
SELECT item_id
      , EXTRACT (YEAR FROM sales_date) AS Yr
      , EXTRACT (MONTH FROM sales_date) AS Mon
      , SUM (sales)
  FROM Daily_Sales
 GROUP BY 1, 2, 3
 ORDER BY 1, 2, 3;
```

item_id	Yr	Mon	Sum(Sales)
5001	2002	1	53987.47
5001	2002	2	45235.03
5001	2002	3	53028.29
5001	2002	4	47632.64
5001	2002	5	53592.29
5001	2002	6	51825.00
5001	2002	7	50452.64
5001	2002	8	53028.29
5001	2002	9	47841.75
5001	2002	10	74663.46
5001	2002	11	65094.86
5001	2002	12	74116.94
5001	2003	1	57433.45
5001	2003	2	46217.14
5001	2003	3	57013.05
5001	2003	6	55732.95
:	:	:	:

The complete EXPLAIN output of this SQL statement follows:

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.MONTHLY_SALES_JI.
 - 2) Next, we lock TFACT.MONTHLY_SALES_JI for read.
 - 3) We do an all-AMPs RETRIEVE step from TFACT.MONTHLY_SALES_JI by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPS. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with high confidence to be 2,520 rows. The estimated time for this step is 0.04 seconds.
 - 4) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.04 seconds.



Aggregation with an Aggregate Index

List the sales by Year and Month for every item.

```
SELECT      item_id
           ,EXTRACT (YEAR FROM sales_date) AS Yr
           ,EXTRACT (MONTH FROM sales_date) AS Mon
           ,SUM (sales)
FROM        Daily_Sales
GROUP BY   1, 2, 3
ORDER BY   1, 2, 3;
```

- An all-rows scan of aggregate join index is used.
- The aggregate index consists of 2520 rows versus 76,685 rows in base table.

EXPLAIN with an Aggregate Index (Partial Listing)

V2R6.2 EXPLAIN

- :
- 3) We do an all-AMPs RETRIEVE step from TFACT.MONTHLY_SALES_JI by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. **The size of Spool 1 is estimated with high confidence to be 2,520 rows. The estimated time for this step is 0.04 seconds.**
- :

Sparse Join Indexes

Another capability of the join index allows you to index a portion of the table using the WHERE clause in the CREATE JOIN INDEX statement to limit the rows indexed. You can limit the rows that are included in the join index to a subset of the rows in the table based on an SQL query result. This is also referred to as a “Partial Covering” join index.

Any join index, whether simple or aggregate, multi-table or single-table, can be sparse.

Examples of how the Sparse Join Index may be used include:

- Ignore rows that are NULL or are most common
- Index rows whose Quantity < 100
- Index a time segment of the table – rows that relate to this quarter

Customer Benefit

A sparse index can focus on the portion of the table(s) that are most frequently used.

- Reduces the storage requirements for a join index
- Makes access faster since the size of the JI is smaller

Like other index choices, a sparse JI should be chosen to support high frequency queries that require short response times. A sparse JI allows the user to:

- Use only a portion of the columns in the base table.
- Index only the values you want to index.
- Ignore some columns, e.g., nulls, to keep access smaller and faster than before.
- Avoid maintenance costs for updates

When the index size is smaller there is less work to maintain and updates are faster since there are fewer rows to update. A sparse JI contents can be limited by date, location information, customer attributes, or a wide variety of selection criteria combined with AND and OR conditions.

Performance

- Better update performance on the base table when its indexes do not contain the most common value(s)
- Smaller index size
- Improved IO and storage
- Collect statistics on the index even if it is only a single column

Limitations

- Sparse Join Indexes follow the same rules as normal Join Indexes



Sparse Join Indexes

Sparse Join Indexes

- Allows you to index a portion of the table using the WHERE clause in the CREATE JOIN INDEX statement to limit the rows indexed.
- Any join index, whether simple or aggregate, multi-table or single-table, can be created as a sparse index.

Examples of how the Sparse Join Index may be used include:

- Ignore rows that are NULL or are most common
- Index rows whose Quantity < 100
- Index a time segment of the table – rows that relate to this quarter

Benefits

- A sparse index can focus on the portion of the table(s) that are most frequently used.
 - Reduces the storage requirements for a join index
 - Faster to create or build
 - Makes access faster since the size of the Join Index is smaller
 - Better update performance on the base table when its indexes do not contain the most common value(s)

Creating a Sparse Join Index

The facing page contains an example of creating a “Sparse Join Index”. The following EXPLAIN shows that the Sparse Join Index is used.

- ... (Locking steps)
- 3) We do an all-AMPs RETRIEVE step from TFACT.CUST_ORD_SJI by way of an all-rows scan with a condition of ("(TFACT.CUST_ORD_SJI.o_orderstatus = 'O') AND ((EXTRACT(DAY FROM (TFACT.CUST_ORD_SJI.o_orderdate)))= 24) AND ((EXTRACT(MONTH FROM (TFACT.CUST_ORD_SJI.o_orderdate)))= 1))" into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with no confidence to be 676 rows. The estimated time for this step is 0.06 seconds.
- 4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.06 seconds.

This following EXPLAIN shows that the Sparse Join Index is not used.

- ... (Locking steps)
- 4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("(TFACT.O.o_orderstatus = 'O') AND (TFACT.O.o_orderdate = DATE '2007-12-18')") into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with low confidence to be 23 rows. The estimated time for this step is 0.38 seconds.
- 5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of a RowHash match scan, which is joined to TFACT.C by way of a RowHash match scan with no residual conditions. Spool 2 and TFACT.C are joined using a merge join, with a join condition of ("TFACT.C.c_custid = o_custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 23 rows. The estimated time for this step is 0.06 seconds.
- 6) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.44 seconds.

PERM Space Required

The amount of PERM space used by this sparse join index as compared to the full join index is listed below.

TableName	SUM(CurrentPerm)
Cust_Ord_JI	1,044,480
Cust_Ord_SJI	322,048



Creating a Sparse Join Index

```
CREATE JOIN INDEX Cust_Ord_SJI AS
  SELECT      (c_custid, c_Iname),
              (o_orderid, o_orderstatus, o_orderdate)
  FROM        Customer C
  INNER JOIN  Orders O
  ON          c_custid = o_custid
  WHERE        EXTRACT(YEAR FROM o_orderdate) = '2008'
  PRIMARY INDEX (c_custid);
```

In this example, a sparse join index is created just for the year 2008.

```
SELECT      c_custid, c_Iname, o_orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          c_custid = o_custid
WHERE      o_orderdate = '2008-01-24'
AND         o_orderstatus = 'O' ;
```

The join index will be used for this SQL and the EXPLAIN estimated cost is 0.06 seconds.

```
SELECT      c_custid, c_Iname, o_orderdate
FROM        Customer C
INNER JOIN  Orders O
ON          c_custid = o_custid
WHERE      o_orderdate = '2007-12-18'
AND         o_orderstatus = 'O' ;
```

The tables will have to be joined for this SQL and the EXPLAIN estimated cost is 0.44 seconds.

Explains are provided on facing page in the PDF file.

Creating a Sparse Join Index on Partitions

The facing page contains an example of creating a sparse join index on a partitioned table. The Sparse Join is only 130,560 bytes in size since it is only created for 3 months.

The following EXPLAIN shows the creation of the sparse join index on a set of partitions.

- : (Locking Steps)
 - 5) We execute the following steps in parallel.
 - 1) We do a single-AMP ABORT test from DBC.DBBase 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.TVFields (no lock required).
 - 8) We do an INSERT into DBC.Indexes (no lock required).
 - 9) We do an INSERT into DBC.TVM (no lock required).
 - 6) We create the table header.
 - 7) We create the index subtable on TFACT.Orders_PPI.
 - 8) We lock DBC.TVM for write on a RowHash, and we lock DBC.Indexes for write on a RowHash.
 - 9) We execute the following steps in parallel.
 - 1) We do an INSERT into DBC.Indexes.
 - 2) We do an INSERT into DBC.Indexes.
 - 3) We do an INSERT into DBC.Indexes.
 - 4) We do an INSERT into DBC.Indexes.
 - 5) We do an INSERT into DBC.Indexes.
 - 6) We do a single-AMP UPDATE from DBC.TVM by way of the unique primary index with no residual conditions.
 - 7) **We do an all-AMPs RETRIEVE step from 3 partitions of TFACT.Orders_PPI with a condition of ("(TFACT.Orders_PPI.o_orderdate <= DATE '2008-03-01') AND (TFACT.Orders_PPI.o_orderdate >= DATE '2008-01-31')") into Spool 1 (all_amps), which is redistributed by hash code to all AMPS. Then we do a SORT to order Spool 1 by row hash. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with high confidence to be 3,983 rows. The estimated time for this step is 0.05 seconds.**
 - 10) We do an all-AMPs MERGE into TFACT.Orders_PPI_JI from Spool 1 (Last Use).
 - 11) We lock a distinct TFACT."pseudo table" for exclusive use on a RowHash to prevent global deadlock for TFACT.Orders_PPI.
 - 12) We lock TFACT.Orders_PPI for exclusive use.
 - 13) We modify the table header TFACT.Orders_PPI and update the table's version number.
 - 14) We lock DBC.AccessRights for write on a RowHash.
 - 15) We INSERT default rights to DBC.AccessRights for TFACT.Orders_PPI_JI.
 - 16) We spoil the parser's dictionary cache for the table.
 - 17) We spoil the parser's dictionary cache for the table.
 - 18) 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.



Creating a Sparse Join Index on Partitions

If the base table is partitioned, a sparse join index may be created for a partition or partitions – only the “partitions of interest” are scanned.

- Creation time of sparse join index is faster because of partition elimination.
- Partition scan of base table instead of full table scan.

```
CREATE SET TABLE Orders_PPI
  ( o_orderid      INTEGER NOT NULL,
    o_custid        INTEGER NOT NULL,
    :
    o_orderdate     DATE FORMAT 'YYYY-MM-DD' NOT NULL,
    :
    o_comment       VARCHAR(79))
PRIMARY INDEX (o_orderid)
PARTITION BY RANGE_N
  ( o_orderdate BETWEEN DATE '2000-01-01' AND DATE '2008-12-31' EACH INTERVAL '1' MONTH );
```

Orders_PPI table is partitioned by month.

```
CREATE JOIN INDEX Orders_PPI_JI AS
  SELECT      o_orderid, o_custid, o_orderstatus, o_totalprice, o_orderdate
  FROM        Orders_PPI
  WHERE       o_orderdate BETWEEN '2008-01-01' AND '2008-03-31'
  PRIMARY INDEX (o_custid);
```

The sparse join index is created for the 1st quarter of 2008.

- 7) We do an all-AMPs RETRIEVE step from 3 partitions of TFACT.Orders_PPI with a condition of ("(TFACT.Orders_PPI.o_orderdate <= DATE '2008-03-31') AND (TFACT.Orders_PPI.o_orderdate >= DATE '2008-01-01')") into Spool 1 (all_amps), ...

3 partitions are scanned to build the JI.

Global (Join) Indexes

A Global Index is a term used to define a join index that contains the Row IDs of the base table rows. Some queries are satisfied by examining only the join index when all referenced columns are stored in the index. Such queries are said to be covered by the join index.

Other queries may use the join index to qualify a few rows, then refer to the base tables to obtain requested columns that aren't stored in the join index. Such queries are said to be partially-covered by the index. This is referred to as a partially-covered global index.

Because the Teradata Database supports multi-table, partially-covering join indexes, all types of join indexes, except the aggregate join index, can be joined to their base tables to retrieve columns that are referenced by a query but are not stored in the join index. Aggregate join indexes can be defined for commonly-used aggregation queries.

A partial-covering join index takes less space than a covering join index, but in general may not improve query performance by as much. Not all columns that are involved in a query selection condition have to be stored in a partial-covering join index. The benefits are:

- Disk storage space for the JI decreases when fewer columns are stored in the JI.
- Performance increases when the number of selection conditions that can be evaluated on the join index increases.

When a Row ID is included in a Join Index, 10 bytes are used for the Row ID (Part # + Row Hash + Uniqueness Value). This is true even if the base table is not partitioned.

Another use for a Global Join Index for a single table is that of a Hashed NUSI. This capability will be described in more detail later in the module.

Customer Benefit

Partial-Covering Global Join Indexes can provide:

- Improved performance for certain class of queries.
- Some of the query improvement benefits that join indexes offer without having to replicate all the columns required to cover the queries resulting in better performance.
- Improved scalability for certain class of queries

Limitations

- Aggregate JI does not support the partial-covering capability.
- Global index is not used for correlated sub-queries.
- Global index is not supported with FastLoad or MultiLoad.



Global (Join) Indexes

A Global Index is a term used to define a join index that contains the Row IDs of all of the tables in the join index.

Example:

- Assume that you are joining 2 tables (Table_A and Table_B) and each has 100 columns. **The join index can include (at most) 64 columns from each base table.**
- You can include the ROWID as part of the 64 columns for each table (ex., A.ROWID and B.ROWID). Each join index subtable row will include the Row IDs of the corresponding base table rows for Table_A and Table_B.
- **The optimizer can build a plan that uses the join index to get most of the data and can join back to either or both of the tables for the rest of the data.**

Another option – use a single table Global Index as a “**Hashed NUSI**” – the join index contains the “secondary index column” and the Row IDs in the join index.

- Useful when a column is used as a secondary index, but the typical number of rows for a value is much less than the number of AMPs in the system.
- For queries with an equality condition on a fairly unique column, it changes:
 - Table-level locks to row hash locks
 - All-AMP steps to group-AMP steps

Global Index – Multiple Tables

The facing page contains an example of creating a “global” multi-table join index.

The total number of columns in the Customer and Orders table is less than 64. Therefore, you could create a join index that includes all of the columns (16 in this case) from the two tables. However, if one table had 70 columns and the other table had 90 columns, you can only include a maximum of 64 columns from each table. Since the number of columns is more than 64, you would include the most frequently referenced columns from each table. Another reason to only include a subset of columns may be to minimize the size of the join index.

In this example, the join index has the most frequently referenced columns as well as the Row IDs of both tables. The join index subtable row will include the Row ID of the base table row for the Customer table and the Row ID of the base table for the Orders table. The optimizer may choose to use the join index to get most of the data and join back to either the Customer or the Orders table. Starting with V2R5, the optimizer can build execution plans that can join back to either table.

The terminology used in EXPLAIN plans that indicates a join back is “... using a row id join ...”.

Join back simply means that the ROWID is carried as part of the join index. This permits the index to “join back” to the base row, much like a NUSI does. It is one of the features of Partial-Covering Join Indexes.

```
EXPLAIN SELECT      c_custid, c_Iname,
FROM                  c_address, c_city, c_state,
INNER JOIN           o_orderdate
ON                      Customer C
WHERE                 Orders O
ORDER BY               c_custid = o_custid
                           o_orderstatus = 'O'
                           1;
```

This EXPLAIN shows that a Global Join Index is used.

- 4) We do an all-AMPs RETRIEVE step from TFACT.CUST_ORD_GJI by way of an all-rows scan with a condition of ("TFACT.CUST_ORD_GJI.o_orderstatus = 'O'") into Spool 2 (all_amps), which is built locally on the AMPS. Then we do a SORT to order Spool 2 by the sort key in spool field1. The size of Spool 2 is estimated with no confidence to be 482 rows. The estimated time for this step is 0.03 seconds.
- 5) We do an all-AMPs JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.C by way of an all-rows scan with no residual conditions. **Spool 2 and TFACT.C are joined using a row id join**, with a join condition of ("Field_1 = TFACT.C.RowID"). The result goes into Spool 1 (group_amps), which is built locally on the AMPS. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with no confidence to be 482 rows. The estimated time for this step is 0.06 seconds.



Global Index – Multiple Tables

```
CREATE SET TABLE Customer
( c_custid      INTEGER NOT NULL,
  c_lname       VARCHAR(15),
  c_fname       VARCHAR(10),
  c_address     VARCHAR(50),
  c_city        VARCHAR(20),
  c_state       CHAR(2),
  c_zipcode     INTEGER)
UNIQUE PRIMARY INDEX ( c_custid );
```

```
CREATE SET TABLE Orders
( o_orderid     INTEGER NOT NULL,
  o_custid      INTEGER NOT NULL,
  o_orderstatus  CHAR(1),
  o_totalprice   DECIMAL(9,2) NOT NULL,
  o_orderdate    DATE
                FORMAT 'YYYY-MM-DD' NOT NULL,
  o_orderpriority SMALLINT,
  o_clerk        CHAR(16),
  o_shippriority SMALLINT,
  o_comment      VARCHAR(79))
UNIQUE PRIMARY INDEX ( o_orderid );
```

```
CREATE JOIN INDEX Cust_Ord_GJI AS
  SELECT          (c_custid, c_lname, C.ROWID AS crid),
                  (o_orderid, o_orderstatus, o_orderdate, O.ROWID AS orid)
  FROM            Customer C
  INNER JOIN      Orders O
  ON              c_custid = o_custid
  PRIMARY INDEX   (c_custid);
```

- The Global Index contains those columns most frequently used in queries – effectively used as “covering join index”.
- The Global Index may be used to join back (via the Row ID) to the tables when columns are referenced that are not part of the join index.

Global Index as a “Hashed NUSI”

Another use for a Global Join Index for a single table is that of a Hashed NUSI. An actual NUSI accesses all AMPs, whereas this index only accesses the AMPs that have rows matching the value. The Global Join Index is hashed, and the system uses the hash to access a single AMP, and then uses the Row IDs in the subtable row to then access only those AMPs that have rows.

ODS (Operational Data Store) or tactical queries that involve an equality condition on a fairly unique column can use a global index which will change:

- Table-level locks to row hash locks
- All-AMP steps to group-AMP steps

Using a global index as a “Hashed NUSI” is similar to a single-table join index with one clear differentiation – it carries a pointer (Row ID) back to the base table, and is used as an alternative means to get to the base table row. It is not used to satisfy a query by itself.

A “hashed NUSI” global index offers the advantages of a NUSI (it supports duplicate rows per value) combined with the advantages of a USI (its index rows are hash-distributed on the indexed value) and is often able to offer group AMP capabilities. As with all Teradata join indexes, its use is transparent to the query and will be determined by the optimizer.

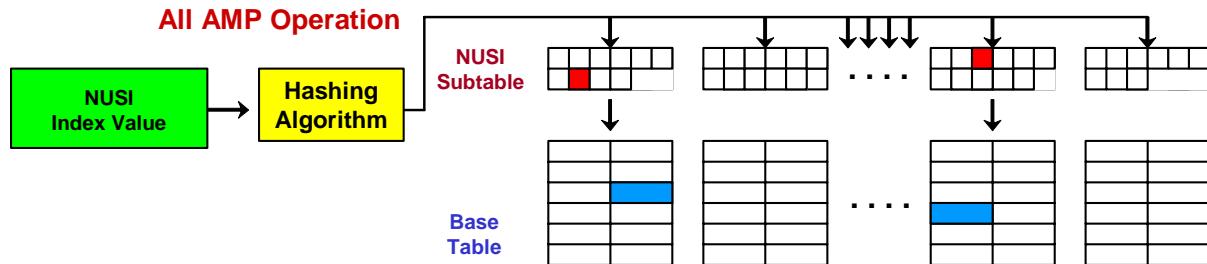
Its main benefit is for situations where you are only getting a few rows for one value, and you can avoid an all AMP operation that a NUSI always requires. This may not have a huge impact on a system with a modest number of AMPs. However, for very large systems, with hundreds or thousands of AMPs, a group AMP operation that engages a small percentage (e.g., only 1% of the total AMPs or less), when done often enough, may increase overall throughput of the platform, as well as faster query response.



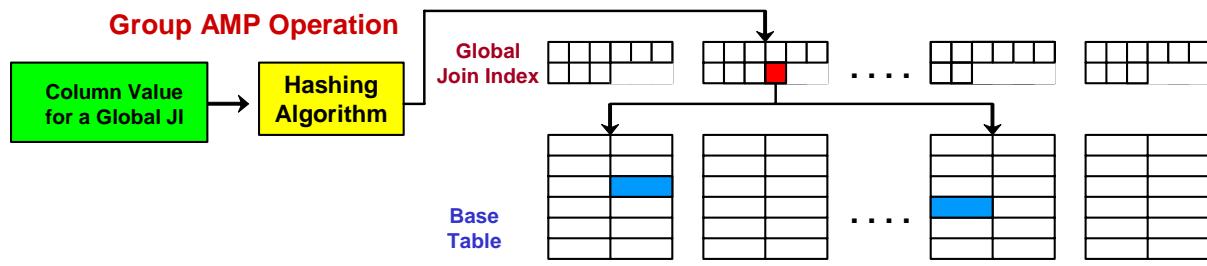
Global Index as a “Hashed NUSI”

Assume 200 AMPs and the typical rows per value is 2 for a column referenced in queries.

- A NUSI on this column avoids a FTS, but it is an **all-AMP operation** and every AMP is required to look in the NUSI subtable for the hash value of the NUSI.



- A Global Join Index (“Hashed NUSI”) on this column utilizes a **Group AMP operation**. The GJI subtable row contains the Row IDs of the base table rows.



Creating a Global Index (“Hashed NUSI”)

The facing page contains an example creating a Global Join Index as a “Hashed NUSI”.

If the number of Row IDs within the Global Index subtable row is less than 50% of the number of AMPs, you will see an EXPLAIN plan with “group-AMPs” operations.

If the number of Row IDs within the Global Index subtable row is more than 50% of the number of AMPs, you will see an EXPLAIN plan with “all -AMPs” operations.

Note: When the Row ID is included in a Join Index, each Row ID is 10 bytes long. The partition number is included even if the base table is not partitioned. The partition number is 0 for non-partitioned tables.

Creating the Global Join Index without “Repeating Row Ids”

If the Global Join Index (that is to be used as a hashed NUSI) is created as follows – without the parenthesis, multiple Row IDs will not be included in a single join index subtable row.

```
CREATE JOIN INDEX Orders_GI2
AS
SELECT          o_custid, ROWID
FROM            Orders
PRIMARY INDEX   (o_custid);
```



Creating a Global Index (“Hashed NUSI”)

```
CREATE JOIN INDEX
  SELECT
    FROM
      PRIMARY INDEX
```

```
Orders_GI AS
  (o_custid),
  (ROWID)
  Orders
  (o_custid);
```

Fixed portion of Join Index contains index value

Repeating portion of Join Index contains Row IDs

One Global Index row can contain multiple Row IDs.

- If the typical rows for a value is less than 50% of the number of AMPs, this global index will yield performance gains.

This effectively means that the number of Row IDs in the subtable row is less than 50% of the number of AMPs.

- EXPLAIN plan will indicate “group-AMPs” operation and “row hash locks”.

- If the typical rows for a value is 50% or greater than the number of AMPs, this global index will result in an all-AMP operation (like a NUSI).

This effectively means that the number of Row IDs in the subtable row is 50% or more than the number of AMPs.

- EXPLAIN plan will indicate “all-AMPs” operation and “table level locks”.

Example: Using a Global Index as a Hashed NUSI

The EXPLAIN of the SQL statement on the facing page is shown below.

- : (locking steps)
 - 3) We do a single-AMP RETRIEVE step from TFACT.ORDERS_GI by way of the primary index "TFACT.ORDERS_GI.o_custid = 1500" with no residual conditions into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by the sort key in spool field1. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 2 is estimated with high confidence to be 14 rows. The estimated time for this step is 0.00 seconds.
 - 4) We do an all-AMPS JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.Orders by way of an all-rows scan with no residual conditions. Spool 2 and TFACT.Orders are joined using a row id join, with a join condition of ("Field_1 = TFACT.Orders.RowID"). The input table TFACT.Orders will not be cached in memory. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 14 rows. The estimated time for this step is 0.08 seconds.
 - 5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.08 seconds.



Example: Using a Global Index as a Hashed NUSI

List the orders for a customer id of 1500.

```
SELECT      o_custid, o_orderid, o_orderstatus, o_orderdate, o_totalprice
FROM        Orders
WHERE       o_custid = 1500;
```

<u>o_custid</u>	<u>o_orderid</u>	<u>o_orderstatus</u>	<u>o_orderdate</u>	<u>o_totalprice</u>
1500	132400	C	2006-08-15	1150.00
1500	136237	C	2006-11-31	2149.00
1500	141842	O	2006-12-18	1207.50

EXPLAIN showing use of a Global Join Index as a Hashed NUSI (Partial Listing)

V2R6.2 EXPLAIN

- :
 3) We do a single-AMP RETRIEVE step from TFACT.Orders_GI by way of the primary index "TFACT.Orders_GI.o_custid = 1500" with no residual conditions into Spool 2 (group_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by the sort key in spool field1. The size of Spool 2 is estimated with high confidence to be 14 rows. The estimated time for this step is 0.00 seconds.
 4) We do a group-AMPS JOIN step from Spool 2 (Last Use) by way of an all-rows scan, which is joined to TFACT.Orders. Spool 2 and TFACT.Orders are joined using a row id join, with a join condition of ("Field_1 = TFACT.Orders.RowID"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with index join confidence to be 4 rows. The estimated time for this step is 0.08 seconds.
 :
 :

Repeating Row Ids in Global Index

The SQL to create a Global Join Index with and without repeating Row IDs is shown on the facing page.

PERM Space Required

The amount of PERM space used by these global join indexes is listed below. Remember that these tables are quite small. Note that the global join index with parenthesis requires less space.

```
SELECT      TableName , SUM(CurrentPerm)
FROM        DBC.TableSize
WHERE       DatabaseName = USER
GROUP BY   1
ORDER BY   1;
```

<u>TableName</u>	<u>SUM(CurrentPerm)</u>	
Orders_GI	1,065,984	(with repeating Row IDs)
Orders_GI2	2,019,840	(without repeating Row IDs)



Repeating Row IDs in Global Index

What is the difference in creating a “Hashed NUSI” with or without repeating Row IDs? **Answer – the amount of PERM space needed.**

```
CREATE JOIN INDEX Orders_GI AS
  SELECT          (o_custid), (ROWID)
    FROM          Orders
   PRIMARY INDEX (o_custid);
```

Global Join Index					
Join Index Row ID	Index Value	(Base Table) Row IDs (Keys)			
RH	1500	RID1	RID2	RID3	
RH	1501	RID4	RID5		

```
CREATE JOIN INDEX Orders_GI2
  SELECT          o_custid, ROWID
    FROM          Orders
   PRIMARY INDEX (o_custid);
```

Global Join Index					
Join Index Row ID	Index Value	(Base Table) Row ID (Key)			
RH	1500	RID1			
RH	1500		RID2		
RH	1500			RID3	
RH	1501			RID4	
RH	1501				RID5

```
SELECT      TableName,
           SUM(CurrentPerm) AS SumPerm
  FROM        DBC.TableSize
 WHERE       DatabaseName = USER
 GROUP BY    1
 ORDER BY    1;
```

TableName	SumPerm	
Orders_GI	1,065,984	(repeating Row IDs)
Orders_GI2	2,019,840	(w/o repeating Row IDs)

Hash Indexes

Hash Indexes are database objects that are user-defined for the purpose of improving query performance. They are file structures that contain properties of both secondary indexes and join indexes. Hash indexes were first introduced with Teradata V2R4.1.

Hash Indexes have an object type of N. Join Indexes have an object type of I.

The hash index provides a space-efficient index structure that can be hash distributed to AMPs in various ways.

The hash index has been designed to improve query performance in a manner similar to a single-table join index. In particular, you can specify a hash index to:

- Cover columns in a query so that the base table does not need to be accessed.
- Serve as an alternate access path to the base table row.

Example Tables

The same Customer and Orders table definitions are also used with Hash Index examples in this module.

```

CREATE SET TABLE Customer
  (c_custid          INTEGER NOT NULL,
   c_lname           VARCHAR(15),
   c_fname           VARCHAR(10),
   c_address         VARCHAR(50),
   c_city            VARCHAR(20),
   c_state           CHAR(2),
   c_zipcode         INTEGER)
UNIQUE PRIMARY INDEX (c_custid);

CREATE SET TABLE Orders
  (o_orderid         INTEGER NOT NULL,
   o_custid          INTEGER NOT NULL,
   o_orderstatus     CHAR(1),
   o_totalprice      DECIMAL(9,2) NOT NULL,
   o_orderdate       DATE FORMAT 'YYYY-MM-DD' NOT NULL,
   o_orderpriority   SMALLINT,
   o_clerk           CHAR(16),
   o_shippriority    SMALLINT,
   o_comment          VARCHAR(79))
UNIQUE PRIMARY INDEX (o_orderid);

```

Note: Statistics have been collected on the primary index, any join columns, and on all hash indexes in these examples.



Hash Indexes

Hash Indexes may also be used to improve query performance. The hash index provides a space-efficient index structure that can be hash distributed to AMPs in various ways.

Similar to secondary indexes in the following ways:

- Created for a single table only.
- **The CREATE syntax is simple and very similar to a secondary index.**
- May cover a query without access of the base table rows.

Similar to join indexes in the following ways:

- They “pre-locate” joinable rows to a common location.
- The distribution and sequencing of the rows is user specified.
- Very similar to single-table join index.

Unlike join indexes in the following ways:

- **Automatically contains base table PI value as part of hash index subtable row.**
- No aggregation operators are permitted.
- They are always defined on a single table.
- No secondary indexes may be built on the hash index.
- A trigger and a hash index **cannot** exist on a table (even in release V2R6.2) – returns error message #3732. In V2R6.2, a join index and a trigger can be on the same table.

Creating a Hash Index

The CREATE HASH INDEX syntax is:

```
CREATE HASH INDEX hash_index_name [, [NO] FALBACK [PROTECTION ] ]
[ , CHECKSUM=checking_level ]
(column_name [ ... ,column_name])
ON tablename
[ BY (column_name [ ... ,column_name] ) ]
[ ORDER BY sort_specification ] ;
```

where sort_specification is one of the following:

```
VALUES
[ VALUES | HASH (column_name [ ... ,column_name] ) ]
```

Hash Index Definition Rules

There are several key rules which govern the use of the BY and ORDER BY clauses:

- The column(s) specified in the BY clause must be a subset of the columns which make up the hash index.
- When the BY clause is specified, the ORDER BY clause must also be specified. This ORDER BY option can be either VALUES or HASH
- If the ORDER BY HASH option is specified, then the BY option also has to be specified.

General Notes

Additional notes include:

- If the ORDER BY VALUES is used without the BY option, the rows are placed on the same AMP as the base table rows (based on row hash of base table PI). However, the row hash of these hash index rows is the 4 byte binary value of the ORDER BY VALUES column and the rows are in this sequence.
- If the ORDER BY VALUES is used with the BY option and the ORDER BY and BY columns are the same, the rows are placed on the AMP based on the hash value of the BY column. However, the hash value of these rows is the 4 byte binary value of the ORDER BY VALUES column and the rows are in this sequence
- If the ORDER BY VALUES is used with the BY option and the BY columns are different than the ORDER BY VALUES column, the rows are placed on the AMP based on the hash value of the BY columns. However, the hash value of these rows is the 4 byte binary value of the ORDER BY VALUES column and the rows are in this sequence.



Creating a Hash Index

Example 1:

```
CREATE HASH INDEX Orders_HI
  (o_custid,
   o_totalprice,
   o_orderdate)
ON Orders;
```

```
CREATE SET TABLE Orders
  ( o_orderid      INTEGER NOT NULL,
    o_custid       INTEGER NOT NULL,
    o_orderstatus  CHAR(1),
    o_totalprice   DECIMAL(9,2) NOT NULL,
    o_orderdate    DATE
                  FORMAT 'YYYY-MM-DD' NOT NULL,
    o_orderpriority SMALLINT,
    o_clerk        CHAR(16),
    o_shippriority SMALLINT,
    o_comment       VARCHAR(79))
UNIQUE PRIMARY INDEX ( o_orderid );
```

Characteristics of this Hash Index are:

- Hash index subtable rows are **AMP-local to the base table** rows and are in row hash sequence based on base table Primary Index value (e.g., o_orderid).
- Hash index subtable rows contain the specified columns and the base table **Primary Index value** which can be hashed and used to join back to the base row.
- Optionally, the ORDER BY clause may be used to have the rows ordered on each AMP in customer id sequence, rather than by the hash value.
- The Hash Index can be used by the Optimizer as a “**covering index**”.
- A Hash Index has an object type of N.

Hash Index – Example 2

The EXPLAIN of the SQL statement on the facing page is shown below.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.ORDERS_HI2.
- 2) Next, we lock TFACT.ORDERS_HI2 for read.
- 3) We do an all-AMPs RETRIEVE step from TFACT.ORDERS_HI2 with a range constraint of ("(TFACT.ORDERS_HI2.o_orderdate <= DATE '2008-01-31') AND (TFACT.ORDERS_HI2.o_orderdate >= DATE '2008-01-01')") with a residual condition of ("(TFACT.ORDERS_HI2.o_totalprice > 500.00) AND ((TFACT.ORDERS_HI2.o_orderdate <= DATE '2008-01-31') AND (TFACT.ORDERS_HI2.o_orderdate >= DATE '2008-01-01'))") into Spool 1 (group_amps), which is built locally on the AMPs. Then we do a SORT to order Spool 1 by the sort key in spool field1. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of Spool 1 is estimated with no confidence to be 1,005 rows. The estimated time for this step is 0.03 seconds.
- 4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.03 seconds.



Hash Index – Example 2

Example 2:

```
CREATE HASH INDEX Orders_HI2 (o_custid, o_totalprice, o_orderdate)
  ON Orders
  ORDER BY VALUES (o_orderdate);
```

Characteristics of this Hash Index are:

- Hash index subtable rows are **AMP-local to the base table** – the BY option isn't used.
- Hash index subtable rows include specified columns and PI value.
- The hash index rows are **stored in “order date” sequence**, rather than in row hash sequence.
- This can be used as a “covering index” and might be more useful for some “range processing” queries.

List customer and order information for all orders greater than \$500 made in January, 2008.

```
SELECT      o_custid, o_orderid,
            o_totalprice, o_orderdate
FROM        Orders
WHERE       o_orderdate
            BETWEEN '2008-01-01' AND '2008-01-31'
AND         o_totalprice > 500
ORDER BY    2 DESC;
```

This Value-Ordered Hash Index is used by the Optimizer and covers the query.

Hash Index – Example 3

The facing page includes an example of creating a hash index that can be used for joins.

It is not necessary to include the “order id” (`o_orderid`) in the hash index definition. It is included automatically as part of the hash index. If you include the primary index column(s) in the hash index row, Teradata does not include them a second time in the actual subtable row.

```
CREATE HASH INDEX Orders_HI5
  (o_orderid, o_custid, o_orderstatus, o_totalprice, o_orderdate)
ON          Orders
BY           (o_custid)
ORDER BY HASH   (o_custid);
```

The size of the subtable for Orders_HI3 (facing page) and the Orders_HI5 (above) is the same. The ORDER BY VALUES for `o_custid` is also a valid option (Orders_HI4) in this example.

Join Index Alternative Technique

A similar effect can be achieved with a single table join index (**STJI**) by adding an explicit ROWID to the join index definition. The ORDER BY VALUES for `o_custid` is not a valid option with a Join Index in this example.

```
CREATE JOIN INDEX Orders_JI3 AS
SELECT      o_orderid, o_custid, o_orderstatus, o_totalprice, o_orderdate,
                  ROWID
FROM        Orders
PRIMARY INDEX  (o_custid);
```

PERM Space Required

The amount of PERM space used by these indexes is listed below. Remember that these tables are quite small.

```
SELECT      TableName, SUM(CurrentPerm)
FROM        DBC.TableSize
WHERE       DatabaseName = USER
GROUP BY 1
ORDER BY 1;
```

<u>TableName</u>	<u>SUM(CurrentPerm)</u>
Orders_HI2	2,740,224
Orders_HI3	3,606,528
Orders_HI4	3,606,528
Orders_HI5	3,606,528
Orders_JI3	3,028,992



Hash Index – Example 3

A Hash Index can be ordered by value or hash.

Create a hash index to facilitate joins between the “Orders” and “Customer” tables, based on the PK/FK relationship on “customer id”.

```
CREATE HASH INDEX Orders_HI3
    (o_custid, o_orderstatus, o_totalprice, o_orderdate)
ON Orders
BY (o_custid)
ORDER BY HASH (o_custid);
```

```
CREATE HASH INDEX Orders_HI4
    (o_custid, o_orderstatus, o_totalprice, o_orderdate)
ON Orders
BY (o_custid)
ORDER BY VALUES (o_custid);
```

Characteristics of these Hash Indexes are:

- Hash index subtable rows are hash distributed by “o_custid” value.
- The **BY option is required when ORDER BY HASH option is specified.**

Hash Index – Example 3 (cont.)

This EXPLAIN is executed against the tables without a Hash Index.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.O.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.O for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs RETRIEVE step from TFACT.O by way of an all-rows scan with a condition of ("(TFACT.O.o_custid >= 1870) AND ((TFACT.O.o_custid <= 1900) AND (TFACT.O.o_orderstatus = 'O'))") into Spool 2 (all_amps), which is redistributed by hash code to all AMPS. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with low confidence to be 375 rows. The estimated time for this step is 0.39 seconds.
 - 5) We do an all-AMPs JOIN step from TFACT.C by way of a RowHash match scan with a condition of ("(TFACT.C.c_custid <= 1900) AND (TFACT.C.c_custid >= 1870)"), which is joined to Spool 2 (Last Use) by way of a RowHash match scan. TFACT.C and Spool 2 are joined using a merge join, with a join condition of ("TFACT.C.c_custid = o_custid"). The result goes into Spool 1 (group_amps), which is built locally on the AMPS. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with low confidence to be 200 rows. The estimated time for this step is 0.06 seconds.
 - 6) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.45 seconds.

This EXPLAIN plan is executed with a **Hash Index** created on the table.

- 1) First, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.ORDERS_HI4.
 - 2) Next, we lock a distinct TFACT."pseudo table" for read on a RowHash to prevent global deadlock for TFACT.C.
 - 3) We lock TFACT.ORDERS_HI4 for read, and we lock TFACT.C for read.
 - 4) We do an all-AMPs JOIN step from TFACT.C by way of an all-rows scan with a condition of ("(TFACT.C.c_custid <= 1900) AND (TFACT.C.c_custid >= 1870)"), which is joined to TFACT.ORDERS_HI4 with a range constraint of ("(TFACT.ORDERS_HI4.o_custid >= 1870) AND (TFACT.ORDERS_HI4.o_custid <= 1900)") with a residual condition of ("(TFACT.ORDERS_HI4.o_custid >= 1870) AND ((TFACT.ORDERS_HI4.o_custid <= 1900) AND (TFACT.ORDERS_HI4.o_orderstatus = 'O')")"). TFACT.C and TFACT.ORDERS_HI4 are joined using a product join, with a join condition of ("TFACT.C.c_custid = TFACT.ORDERS_HI4.o_custid"). The input table TFACT.ORDERS_HI4 will not be cached in memory, but it is eligible for synchronized scanning. The result goes into Spool 1 (group_amps), which is built locally on the AMPS. Then we do a SORT to order Spool 1 by the sort key in spool field1. The size of Spool 1 is estimated with no confidence to be 267 rows. The estimated time for this step is 0.06 seconds.
 - 5) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.
- > The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.06 seconds.



Hash Index – Example 3 (cont.)

List the customers with customer ids between 1870 and 1900 who have open orders?

```

SELECT      c_custid, c_lname,
            o_orderid, o_orderdate
FROM        Customer C
INNER JOIN Orders O
ON          c_custid = o_custid
WHERE       o_orderstatus = 'O'
AND         c_custid BETWEEN 1870 AND 1900
ORDER BY    1;
  
```

c_custid	c_lname	o_orderid	o_orderdate
1875	Porter	114105	2006-10-02
1876	Hengster	114166	2006-10-02
:	:	:	:

SQL Query	Time
Without Hash Index	.45 seconds
With Hash Index	.06 seconds

- The rows of the customer table and the Hash Index are located on the same AMP.
- This Hash Index utilizes the range constraint within the query.

EXPLAIN using Hash Index (Partial Listing)

V2R6.2 EXPLAIN

- 4) We do an all-AMPs JOIN step from TFACT.C by way of an all-rows scan with a condition of ("(TFACT.C.c_custid <= 1900) AND (TFACT.C.c_custid >= 1870)"), which is joined to TFACT.ORDERS_HI4 with a range constraint of ("(TFACT.ORDERS_HI4.o_custid >= 1870) AND (TFACT.ORDERS_HI4.o_custid <= 1900)") with a residual condition of ("(TFACT.ORDERS_HI4.o_custid >= 1870) AND ((TFACT.ORDERS_HI4.o_custid <= 1900) AND (TFACT.ORDERS_HI4.o_orderstatus = 'O'))").

Summary

The facing page summarizes the key topics presented in this module.



Summary

Teradata provides additional index choices that can be used to improve performance for known queries.

Reasons to use a Join Index:

- May be used to pre-join multiple tables.
- May be used as an aggregate index.
- The WHERE clause can be used to limit the number of rows in the join index.
 - Referred to a “**Sparse Index**”.
- Row ID(s) of table (or tables) can be included to create a “**Global Index**”.
- May be used as a “**Hashed NUSI**”.
- Secondary indexes can be created on a join index. Secondary indexes can be ordered by value or hash.

Reasons to use a Hash index (instead of a Join Index):

- Automatically includes the Primary Index value.
- The syntax is similar to secondary index syntax, thus simpler SQL to code.
- The Hash Index can be ordered by value or hash.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Check the box if the attribute applies to the index.

	Compressed Join Index Syntax	Non- Compressed Join Index Syntax	Aggregate Join Index	Sparse Join Index	Hash Index
May be created on a single table	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
May be created on multiple tables	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Requires the use of SUM or COUNT functions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Requires a WHERE condition to limit rows stored in the index.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Automatically updated as base table rows are inserted or updated	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Automatically includes the base table PI value as part of the index	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Lab Exercise 30-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To populate a table:

```
INSERT      INTO new tablename
SELECT *    FROM existing tablename;
```



Lab Exercises

Lab Exercise 30-1

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to create a join index and evaluate the Explains of various joins.

What you need

Populated PD tables and Employee, Department, and Job tables in your database

Tasks

1. Verify the number of rows in Employee, Job, and Department tables. If not correct, use INSERT/SELECT to populate the tables from the PD database.

Employee Count = 1000
Department Count = 60
Job Count = 66

2. EXPLAIN the following SQL statement.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
FROM        Employee E
INNER JOIN  Department D      ON  E.Dept_Number = D.Dept_Number
INNER JOIN  Job J            ON  E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

What is estimated time cost for this EXPLAIN? _____

Lab Exercise 30-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To create a join index:

```
CREATE JOIN INDEX join_index_name AS  
SELECT .... ;
```

SUM of Perm space using the DBC.TableSize view.

```
SELECT      TableName (CHAR(15)), SUM(CurrentPerm)  
FROM        DBC.TableSize  
WHERE       DatabaseName = DATABASE  
AND         TableName = 'join_index_name'  
GROUP BY    1  
ORDER BY    1;
```



Lab Exercises

Lab Exercise 30-1 (cont.)

3. Create a “non-compressed” join index which includes the following columns of Employee, Department, and Job.

Employee – Last_name, First_Name
Department – Dept_Number, Dept_Name
Job – Job_Code, Job_Desc

Execute the HELP USER command. What is the object type of the Join Index? _____

4. EXPLAIN the following SQL statement (same SQL as step #2)

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc
FROM        Employee E
INNER JOIN  Department D  ON E.Dept_Number = D.Dept_Number
INNER JOIN  Job J         ON E.job_code = J.job_code
ORDER BY    3, 1, 2;
```

What is estimated time cost for this EXPLAIN? _____

Is the join index used? _____

How much space does the join index require (use the DBC.TableSize view)? _____

Lab Exercise 30-1 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.



Lab Exercises

Lab Exercise 30-1 (cont.)

5. EXPLAIN the following SQL statement – Salary_Amount has been added as a projected column.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc, Salary_Amount
FROM        Employee E
INNER JOIN  Department D  ON E.Dept_Number = D.Dept_Number
INNER JOIN  Job J         ON E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

What is estimated time cost for this EXPLAIN? _____

Is the join index used? _____ If not, why not? _____

6. Drop the Join Index.

Teradata Training

Notes

Module 31



Miscellaneous SQL Features

After completing this module, you will be able to:

- State the purpose and function of the session setting flags.
- Recognize differences in transaction modes for Teradata and ANSI.
- Distinguish between ANSI and Teradata case sensitivities.
- Describe 2 features of the System Calendar.
- List the statements that cause a trigger to “fire”.
- Describe how space is allocated for “volatile” and “global” temporary tables.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Teradata SQL	31-4
Who is ANSI?	31-4
SQL – Version 2 Differences	31-6
SQL Session Modes	31-8
Transaction Modes – Teradata	31-10
Transaction Modes – ANSI.....	31-12
Transaction Mode Examples.....	31-14
Multi-Statement Requests	31-16
CASE Sensitivity Issues.....	31-18
Teradata Mode	31-18
ANSI Mode	31-18
Using ANSI Blind Test.....	31-18
Setting the SQL Flagger.....	31-20
SQLFLAG Example	31-22
HELP SESSION Command.....	31-24
BTEQ .SHOW Command.....	31-24
Why a System Calendar?	31-26
Calendar Table Layout.....	31-28
One Row in the Calendar	31-30
Using the Calendar.....	31-32
What is a Trigger?	31-34
Trigger Example	31-36
Trigger Example (cont.).....	31-38
Temporary Table Choices	31-40
Derived Tables Revisited	31-42
Volatile Tables	31-44
Volatile Table Restrictions.....	31-46
Global Temporary Tables	31-48
Creating Global Temporary Tables.....	31-50
V2R5 – New Features	31-52
V2R6 – New Features	31-54
Teradata 12.0 – New Features.....	31-56
Teradata Limits (Different Releases).....	31-58
Review Questions	31-60

Teradata SQL

SQL is a standard, open language without corporate ownership. The commercial acceptance of SQL was precipitated by the formation of SQL Standards committees by the American National Standards Institute and the International Standards Organization in 1986 and 1987. Two years later they published a specification known as SQL-89 (SQL1). An improvement and expansion to the SQL1 standard gave the world SQL-92 (SQL2). We now have the third generation standard, SQL-99 (SQL3). The existence of standards is important for the general portability of SQL statements.

Teradata SQL has evolved from a DB2 compatible syntax under V1 to an ANSI compliant syntax under V2. In both cases Teradata has its own extensions to the language. Current certification is at entry level (SQL2) with some enhanced features implemented.

With the release of Teradata V2R2, Teradata SQL has evolved to include ANSI compliant syntax. Teradata has its own extensions to the language, as do most RDBMS vendors. With V2R2, Teradata is fully certified at the SQL2 Core or Entry Level. Some Enhanced level features have also been implemented as well.

As the Teradata Database evolves, each major release is expected to move Teradata SQL closer to conformance with the enhanced level of SQL-99 (SQL3).

The SQL-99 (SQL3) specification includes Triggers, Stored Procedures, Domain Specifications, and Object-Oriented Support.

Who is ANSI?

The American National Standards Institute is an administrator and coordinator of voluntary systems of standardization for the United States private sector. About 80 years ago a group of engineering societies and government agencies formed the institute to enhance the “quality of life by promoting and facilitating voluntary consensus standards and conformity.” Today the Institute represents the interests of about 1,000 companies, organizations and government agencies. ANSI does not itself develop standards; rather it facilitates development by establishing consensus among qualified groups.

Acronym: NIST – National Institute of Standards and Technology



Teradata SQL

<u>OS</u>	<u>Platform</u>	<u>SQL</u>	<u>Compatibilities</u>
TOS	DBC/1012	DBC/SQL	DB2, SQL/DS, ANSI
TOS	3600	Teradata SQL (V1R5.1)	DB2, SQL/DS, ANSI (Outer Join added)
UNIX MP-RAS	5100	Teradata SQL (V2R1)	DB2, SQL/DS, ANSI (Similar to V1R5.1)
UNIX MP-RAS Windows 2003 Linux	52xx – 55xx 52xx – 55xx 54xx – 55xx	Teradata SQL (V2R2 to Teradata 12.0)	ANSI (major syntax change)

Three ANSI standards:

- ANSI SQL-89 (SQL1)
- ANSI SQL-92 (SQL2)
- ANSI SQL-99 (SQL3)

Two levels of ANSI SQL99 compliance:

- Core level (*CoreSQL99*) - previously Entry level
- Enhanced level (*EnhancedSQL99*)

Teradata SQL(Version 2):

- Teradata ANSI-99 compliant – Core level
- Certified by US Government and NIST
- Includes many of the enhanced features

SQL – Version 2 Differences

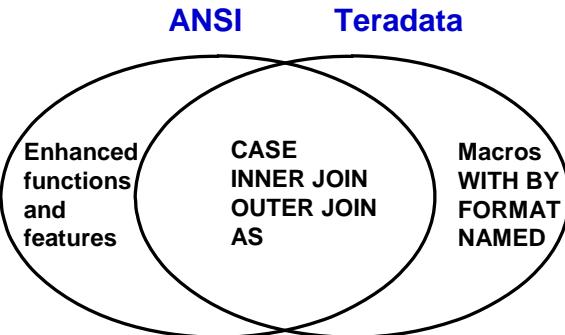
Teradata SQL under V2 contains all existing features of V1, with the addition of new functionality using ANSI syntax and also ANSI compliant ways to implement some of the non-ANSI V2 features.

Version 2 allows for different modes for session operation: ANSI mode and Teradata (BTET) mode. The choice of mode determines the transaction protocol behavior, but also affects such things as case sensitivity defaults, collating sequences, data conversions and display functions. It is important to note that the exact same SQL statement might perform differently in each mode based on these considerations.

Regardless of mode selected, all syntax, whether ANSI compliant or not, is useable. The choice of mode does not inhibit any functionality.



SQL Version 2 Differences



Teradata Version 2 consists of:

- All existing Teradata features from V1.
- ANSI compatible ways to perform many Teradata V1 features.
- Completely new features based on ANSI standard.

V2R2 (and later) allows for sessions to operate in either ...

- **BTET (Teradata) mode**
- **ANSI mode**

All syntax, both ANSI and Teradata extensions, is accepted in either mode.

The same syntax might function differently in each mode.

- Transaction protocol behavior
- CREATE TABLE SET or MULTISET default
- Case sensitivity and collating sequences
- Certain data conversion and display functions

SQL Session Modes

A session flag may be set for the transaction mode of the session. A session in Teradata mode will operate with BEGIN and END TRANSACTION protocols while a session in ANSI mode will operate with COMMIT protocol. There are other subtle differences in each mode's treatment of CREATE TABLE defaults, case sensitivity, collation sequences, data conversion and display.

A comparison summary chart follows:

Teradata Mode	ANSI Mode
The default is that a transaction is implicit . Explicit transactions are available using the BT and ET commands.	All transactions are explicit and a COMMIT WORK is required to successfully commit all completed work.
CREATE TABLE – defaults to SET table	CREATE TABLE – defaults to MULTISET table
Data comparison is NOT case specific.	Data comparison is case specific.
Allows truncation of display data.	Forbids truncation of display data.



SQL Session Modes

Transaction mode setting

BTET – uses standard Teradata mode (Begin Txn – End Txn mode)

ANSI – uses ANSI mode (Commit mode)

BTEQ Examples

.SET SESSION TRANSACTION BTET;

- requires neither for implicit transactions
- requires BT to start explicit transaction
- requires ET to end explicit transaction

.SET SESSION TRANSACTION ANSI;

- requires COMMIT to end transaction

Must be entered prior to LOGON. To change session mode, must LOGOFF first.

Session Mode affects:

- Transaction protocol
- CREATE TABLE defaults
- Default case sensitivities
- Data conversions

Transaction Modes – Teradata

Teradata mode is also referred to as **BTET mode** (Begin Transaction/End Transaction). It is the standard V1 mode wherein all individual requests are treated as single implicit transactions. To aggregate requests into a single transaction requires the BEGIN and END TRANSACTION delimiters.



Transaction Modes – Teradata

.SET SESSION TRANSACTION BTET;

BTET mode characteristics:

- CREATE TABLE default – SET table
- A transaction is by definition implicit.
 - Each request is an implicit transaction.

BT / ET Statements

- BEGIN TRANSACTION (**BT**) and END TRANSACTION (**ET**) statements are used to create larger transactions out of individual requests.
- **BT;** – begins an explicit transaction
- **ET;** – commits the currently active transaction
- Locks are accumulated following a BT until an ET is issued.
- A DDL statement must be the last statement before an ET.
- A rollback occurs when any of the following occur:

– ROLLBACK WORK	- explicit rollback of active Txn
– SQL statement failure	- rollback of active Txn
– Session abort	- rollback of active Txn

Transaction Modes – ANSI

ANSI mode is also referred to as **COMMIT mode**. ANSI mode automatically aggregates requests until an explicit COMMIT command is encountered. Thus, all transactions in ANSI mode are by definition explicit.

Note that all DDL statements must be immediately delimited by a COMMIT and also that macros containing DDL must contain only a single DDL statement and must also be followed by an immediate commit.



Transaction Modes – ANSI

.SET SESSION TRANSACTION ANSI;

ANSI mode characteristics:

- CREATE TABLE default – MULTISET table
- A transaction is committed only by an explicit COMMIT.
 - COMMIT WORK will commit the currently active Txn.
- Transactions are by definition explicit.
- Statement following a COMMIT automatically starts a new Txn.
- A DDL statement must be the last statement before a COMMIT.
- Locks are accumulated until a COMMIT is issued.
- A rollback occurs when any of the following occur:
 - ROLLBACK WORK - explicit rollback of active Txn
 - Session abort - rollback of active Txn
 - SQL statement failure - rollback current statement only

Transaction Mode Examples

The facing page shows the various permutations of transaction modes and the expected results from success, failure and rollback.



Transaction Mode Examples

<u>ANSI Mode</u>	<u>BTET Mode</u> (explicit)	<u>BTET Mode</u> (implicit)
UPDATE A ... ; UPDATE B ... ; COMMIT; (Both commit)	BT; UPDATE A ... ; UPDATE B ... ; ET; (Both commit)	UPDATE A ... ; (A commits) UPDATE B ... ; (B commits)
UPDATE A ... ; UPDATE B ... ; (Fails) COMMIT; (A commits)	BT; UPDATE A ... ; UPDATE B ... ; (Fails) (Both rollback)	UPDATE A ... ; (A commits) UPDATE B ... ; (Fails) (Rollback B)
UPDATE A ... ; UPDATE B ... ; ROLLBACK ; (Both rollback)	BT; UPDATE A ... ; UPDATE B ... ; ROLLBACK ; (Both rollback)	(No explicit ROLLBACK in implicit Txn)
UPDATE A ... ; UPDATE B ... ; LOGOFF; (Both rollback)	BT; UPDATE A ... ; UPDATE B ... ; LOGOFF; (Both rollback)	UPDATE A ... ; (A commits) UPDATE B ... ; (B commits) LOGOFF;

Multi-Statement Requests

A multi-statement DML request is shown on the facing page. A semicolon at the end of a line defines the end of the request. These three UPDATE statements will be executed in parallel.

As described on the facing page, requests have locks acquired up front in ascending TableID order which minimizes the chance of deadlocks if the same request is executed by other users or if other requests using the same tables are executed.

The term **request** is used to refer to any of the following:

- A multi-statement request. Used only with DML (Data Manipulation Language) requests.
- A single statement request. Used with DDL (Data Definition Language) or DML requests.
- A macro. Used with multi-statement or single statement requests, following the above rules.

The three types of requests above are also considered "implicit transactions" (or "system-generated" transactions). In fact, it is because these requests are transactions that their locks are held until the requests complete.

If locks are placed in separate requests, their order will be defined by the order of the requests. This is not recommended since this order may be different than the order that would be used in a single request. To prevent deadlocks, it is helpful to place all locks at the beginning of a transaction in a single request (especially for database and table-level locks).



Multi-Statement Requests

```
UPDATE Dept          SET Salary_Change_Date = CURRENT_DATE  
; UPDATE Manager    SET Salary_Amt = Salary_Amt * 1.06  
; UPDATE Employee   SET Salary_Amt = Salary_Amt * 1.04 ;
```

This is an example of 1 request – 3 statements. This one request is considered an “[implicit transaction](#)”.

Notes:

- A semi-colon at the end of a line defines the end of the request (BTEQ convention).
- You cannot mix DDL and DML within a single request.

The 3 table-level write locks (in this example) will be:

- Acquired in TID order.
- Held until done.

Advantage: Minimizes deadlocks at the table level when many users execute requests on the same tables.

This applies for all types of requests:

- Multi-statement requests (as above)
- Single-statement DDL or DML requests
- Macros

CASE Sensitivity Issues

Teradata Mode

Teradata mode uses the same defaults that are familiar to the V1 user. Data is stored as entered unless an UPPERCASE attribute is specified for the column.

The default for character comparisons is NON-CASESPECIFIC unless either the CASESPECIFIC or UPPER/LOWER operators are specified as part of the comparison criteria.

ANSI Mode

ANSI mode always stores data as entered. The default mode for data comparison is always CASESPECIFIC unless the UPPER/LOWER operator is used as part of the comparison criteria.

Note the use of CASESPECIFIC and NONCASESPECIFIC operators are non-ANSI compliant syntax.

Using ANSI Blind Test

Because ANSI does not permit use of CASESPECIFIC and NONCASESPECIFIC as comparison operators or as column attributes, ANSI provides the UPPER operator as a means for doing a “case-blind” comparison of characters. Using this technique will allow a script to function compatibly in either ANSI or Teradata mode.

Teradata Mode

```
SELECT first_name, last_name
FROM Employee
WHERE last_name LIKE '%Ra%';
```

first_name	last_name
Robert	Crane
James	Trader
I.B.	Trainer
Larry	Ratzlaff
Peter	Rabbit

ANSI Mode

```
SELECT first_name, last_name
FROM Employee
WHERE UPPER(last_name)
      LIKE UPPER('%Ra%');
```

first_name	last_name
Robert	Crane
James	Trader
I.B.	Trainer
Larry	Ratzlaff
Peter	Rabbit



CASE Sensitivity Issues

Column Attributes	Teradata Mode	ANSI Mode
Storage	As entered (default) UPPERCASE	None (As entered is default)
Comparisons	UPPER, LOWER CS (Case specific) NOT CS (Not case specific – Teradata Default)	UPPER, LOWER (Case specific is ANSI default)

Teradata Mode – Default is NOT CS

```
SELECT first_name, last_name
FROM Employee
WHERE last_name LIKE '%Ra%';

first_name    last_name
-----        -----
Robert        Crane
James         Trader
I.B.          Trainer
Larry         Ratzlaff
Peter         Rabbit
```

ANSI Mode – Default is “case specific”

```
SELECT first_name, last_name
FROM Employee
WHERE last_name LIKE '%Ra%';

first_name    last_name
-----        -----
Larry         Ratzlaff
Peter         Rabbit
```

ANSI Blind Test – an example of executing a “non case specific” compare in ANSI mode is provided on the facing page.

Setting the SQL Flagger

An additional BTEQ setting is available to affect the session mode. An SQLFLAG may be enabled to flag any syntax which is non-ANSI compliant. This flag does not inhibit the execution of any commands; rather it generates warning when any ANSI non-compliance is detected.

It is possible to use the following syntax to see ANSI SQL-99 Enhanced violations. However, this setting does not show core violations and has little meaning or use with Teradata.

.SET SESSION SGLFLAG INTERMEDIATE



Setting the SQL Flagger

Teradata sessions have an additional selectable attribute to flag ANSI SQL non-compliance.

SQLFLAG setting

- | | |
|-------|-------------------------------------|
| ENTRY | – flags ANSI core incompatibilities |
| NONE | – turns off flagger |

BTEQ Example

```
.SET SESSION SQLFLAG ENTRY; – flags non-core level ANSI syntax
```

Must be entered prior to LOGON. To change session mode, must LOGOFF first.

- Affects:
- Warnings generated for ANSI non-compliance
 - No effect on command execution

For example:

DATE is not ANSI standard. CURRENT_DATE is ANSI standard.

SQLFLAG Example

An example is shown of warnings generated by the SQLFlagger for a single SQL statement to select today's date. Note that following the warnings, the date is returned.

The following error codes are from the Teradata Messages manual.

5836 Token is not an entry level ANSI Identifier or Keyword.

Explanation: An identifier or keyword is not compliant with entry level ANSI rules.

Generated By: LEXER.

For Whom: User.

Remedy: If script is to be full ANSI compliant, change the indicated statement.

Note: This error is given because SELECT must be in uppercase.

5818 Synonyms for Operators or Keywords are not ANSI.

Explanation: A non-ANSI synonym has been used for a Keyword or Operator.

Generated By: SYN modules

For Whom: User.

Remedy: If script is to be full ANSI compliant, change the indicated statement.

Note: This error is given because SELECT must be fully spelled out.

5821 Built-in values DATE and TIME are not ANSI.

Explanation: These values are not supported in ANSI.

Generated By: SYN modules.

For Whom: User.

Remedy: If script is to be full ANSI compliant, change the indicated statement.

Note: This error is given because CURRENT_DATE must be used and in uppercase.

5804 A FROM clause is required in ANSI Query Specification.

Explanation: A query has been submitted that does not include a FROM clause.

Generated By: SYN modules.

For Whom: User.

Remedy: If script is to be full ANSI compliant, change the indicated statement.



SQLFLAG Example

```
.set session sqlflag entry;
.logon tfact03,tfact03;
sel date;

*** Query completed. One row found.
*** One column returned.
*** Total elapsed time was 1 second.

sel date;
$
*** SQL Warning 5836 Token is not an entry level ANSI
Identifier or Keyword.

sel date;
$
*** SQL Warning 5818 Synonyms for Operators or Keywords
are not ANSI.

sel date;
$
*** SQL Warning 5821 Built-in values DATE and TIME are not ANSI.

sel date;
$
*** SQL Warning 5804 A FROM clause is required in ANSI Query Specification.
```

```
.logoff
.set session sqlflag none;
.logon tfact03,tfact03;
sel date;

*** Query completed. One row found.
*** One column returned.
*** Total elapsed time was 1 second.
```

Current Date

2008-01-21

HELP SESSION Command

There are new HELP features available to the user of Teradata SQL under V2.

Help at the session level shows whether Teradata (BTET) mode or COMMIT (ANSI) mode are invoked for the session.

BTEQ .SHOW Command

The **BTEQ .SHOW command** shows all settings enabled for a BTEQ invoked session of the Teradata DBC. Because BTEQ is primarily a client utility for report generation, many of the settings are reporting specifications. There are other settings that reflect BTEQ's import and export features as well.

The SHOW command displays session settings including the ANSI Flagger and the specified transaction mode.

```
.SHOW CONTROL

:
[SET] SEPARATOR = two blanks
[SET] SESSION CHARSET = ASCII
[SET] SESSION RESPBUFLEN = 8192
[SET] SESSION SQLFLAG = NONE
[SET] SESSION TRANSACTION = BTET
[SET] SESSION TWORESPBUFS = ON
:
```



HELP SESSION Command

HELP SESSION;

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

User Name	TFACT03
Account Name	\$M_&S_&D&H
Logon Date	08/01/21
Logon Time	11:26:44
Current DataBase	TFACT
Collation	ASCII
Character Set	ASCII
Transaction Semantics	Teradata
Current DateForm	IntegerDate
Session Time Zone	00:00
Default Character Type	LATIN
:	:
Default Date Format	YY/MM/DD
:	:
Currency Name	US Dollars
Currency	\$
:	:
Default Timestamp format	YYYY-MM-DDBHH:MI:SS.S(F)Z
Current Role	TF_Student
Logon Account	\$M_&S_&D&H
Profile	Student_P
LDAP	N
Audit Trail ID	TFACT03
Queryband	

BTEQ Note: To produce this format in BTEQ, use these BTEQ settings:

.SET SIDETITLES
 .SET FOLDLINE

To return to the default settings:

.SET DEFAULTS

Notes:

- The TD 12.0 HELP SESSION displays more parameters than previous releases.
- However, to see SQLFLAGGER setting, use SHOW CONTROL command.

Why a System Calendar?

Structured Query Language (SQL) permits a certain amount of mathematical manipulation of dates, however the needs of real world applications often exceed this innate capability. Implementing a system calendar is often necessary to answer time-relative business questions. Summarizing information based on a quarter of the year or on a specific day of the week can be onerous without the assistance of a system calendar.

As implemented for Teradata, the **System Calendar** is a high-performance set of nested views which, when executed, materialize date information as a dimension in a star schema. The system calendar is easily joined to other tables to produce information based on any type of time period or time boundary.

The underlying base table consists of one row for each day within the range of Jan 1, 1900 through Dec. 31, 2100. There is only one column, a date, in each row. Each level of view built on top of the base table adds intelligence to the date.



Why A System Calendar?

The Truth Is ...

SQL has limited ability to do date arithmetic.
There is a need for more complex, calendar-based calculations.

I'd Like To Know ...

*How does this quarter compare to same quarter last year?
How many shoes do we sell on Sundays vs. Saturdays?
During which week of the month do we sell the most pizzas?*

Some Good News

Extends properties of DATE data type by joining to Calendar.
Easily joined to other tables, i.e., dimension of a star schema.
High performance - limited I/O.
Has advantages over user-defined calendars.

Standard Usage

Statistics are created for materialized table for join planning.
Only necessary rows are materialized for the calendar.

Calendar Table Layout

The views and the base table that make up the system calendar are contained in a database called ‘Sys_Calendar’. The contents of this database are easily seen with the help of the HELP DATABASE command.

HELP DATABASE Sys_Calendar;

*** Help information returned. 4 rows.
 *** Total elapsed time was 1 second.

<u>Table/View/Macro name</u>	<u>Kind</u>	<u>Comment</u>
CALENDAR	V	?
CALENDARTMP	V	?
CALBASICS	V	?
CALDATES	T	?

The base table for the system calendar contains a row for each date between Jan 1, 1900 through Dec 31, 2100. Each row contains a single column that is a DATE data type. This is demonstrated using the SHOW TABLE command.

SHOW TABLE Sys_Calendar.Caldates;

*** Text of DDL statement returned.
 *** Total elapsed time was 1 second.

```
CREATE SET TABLE Sys_Calendar.Caldates, Fallback ,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL
(
  cdate DATE FORMAT 'YY/MM/DD')
UNIQUE PRIMARY INDEX ( cdate );
```



Calendar Table Layout

Columns from the System Calendar:

calendar_date DATE UNIQUE (Standard Teradata date)
 day_of_week BYTEINT, (1-7, where 1 = Sunday)
 day_of_month BYTEINT, (1-31)
 day_of_year SMALLINT, (1-366)
 day_of_calendar INTEGER, (Julian days since 01/01/1900)
 weekday_of_month BYTEINT, (nth occurrence of day in month)
 week_of_month BYTEINT, (partial week at start of month is 0)
 week_of_year BYTEINT, (0-53) (partial week at start of year is 0)
 week_of_calendar INTEGER, (0-n) (partial week at start is 0)
 month_of_quarter BYTEINT, (1-3)
 month_of_year BYTEINT, (1-12)
 month_of_calendar INTEGER, (1-n) (Starting Jan, 1900)
 quarter_of_year BYTEINT, (1-4)
 quarter_of_calendar INTEGER, (Starting Q1, 1900)
 year_of_calendar SMALLINT, (Starting 1900)

System Calendar is a 4-level nested view of dates.

Underlying table is Sys_calendar.Caldates:

- Has one column called 'cdate' - DATE data type.
- Has one row for each date of calendar.
- Unique Primary Index is cdate.
- Each level of view adds intelligence to date.

Note:

System calendar includes
Jan 1, 1900 through
Dec. 31, 2100.

One Row in the Calendar

Four Levels of Calendar Views

Calendar [View which views Calendartmp](#)

Calendartmp [View which adds:](#)
 day_of_week
 weekday_of_month
 week_of_month
 week_of_year
 week_of_calendar
 month_of_quarter
 month_of_calendar
 quarter_of_year
 quarter_of_calendar

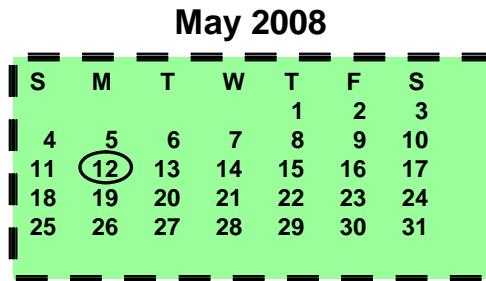
Calbasics [View which adds:](#)
 calendar_date,
 day_of_calendar,
 day_of_month,
 day_of_year,
 month_of_year,
 year_of_calendar)

Caldates [Underlying table which contains a date column:](#)

One Row in the Calendar

```
SELECT * FROM Sys_Calendar.Calendar WHERE calendar_date = '2008-05-12';
```

calendar_date	2008-05-12
day_of_week	2
day_of_month	12
day_of_year	133
day_of_calendar	39579
weekday_of_month	2
week_of_month	2
week_of_year	19
week_of_calendar	5654
month_of_quarter	2
month_of_year	5
month_of_calendar	1301
quarter_of_year	2
quarter_of_calendar	434
year_of_calendar	2008



Note: `SELECT CURRENT_DATE` is the ANSI standard equivalent of `SELECT DATE`.

Using the Calendar

The daily sales table is used in the following example:

```
CREATE SET TABLE Daily_Sales,
    NO FALBACK,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
        ( itemid      INTEGER,
          salesdate   DATE FORMAT 'YYYY/MM/DD',
          sales       DECIMAL(9,2) )
PRIMARY INDEX ( itemid );
```

The following is a non-ANSI standard way of performing the join on the facing page.

```
SELECT      DS.itemid
            ,SUM(DS.sales)
FROM        daily_sales           DS,
            Sys_Calendar.Calendar SC
WHERE       DS.salesdate = SC.calendar_date
AND         SC.quarter_of_year = 1
AND         SC.year_of_calendar = 2008
AND         DS.itemid = 10
GROUP BY 1
;
```



Using the Calendar

Show total sales of item 10 reported in Q1 of 2008.

Daily_Sales table

Item_id
salesdate
sales

Sys_Calendar.Calendar

calendar_date
:
:
quarter_of_year
:
year-of_calendar

= 1 ?

= 2008 ?

SQL:

```
SELECT      DS.itemid, SUM(DS.sales)
FROM        Daily_Sales          DS
INNER JOIN  Sys_Calendar.Calendar SC
ON          DS.salesdate = SC.calendar_date
AND         SC.quarter_of_year = 1
AND         SC.year_of_calendar = 2008
AND         DS.itemid = 10
GROUP BY    1;
```

Result:

itemid	Sum(sales)
10	4050.00

Salesdate is joined to the system calendar.
Calendar determines if this date meets this criteria:

Is it a Quarter 1 date?
Is it a 2008 date?
If yes on both, add this sales amount to result.

What is a Trigger?

A **trigger** is an object in a database, like a macro or view is an object in a database. A trigger is created with a CREATE TRIGGER statement and defines events that happen when some other event, called a *triggering event*, occurs.

A **trigger** consists of one or more SQL statements that are associated with a table and which are executed when the trigger is “fired”.

Limitations on triggers include the inability to use FastLoad, MultiLoad or Updateable Cursors on tables for which triggers are defined.

Much of the standard DDL that applies to other database objects also applies to triggers.

What is a Trigger?

A Trigger may be defined as:

- One or more stored SQL statements associated with a table
- An event driven procedure attached to a table
- An object in a database, like tables, views and macros

Triggers may **not** be used in conjunction with:

- The FastLoad utility
- The MultiLoad utility
- Updateable Cursors

Any of the following SQL statements may be applied to triggers:



DELETE DATABASE or **DELETE USER** – cause all triggers to be dropped.

Privileges are required to create and drop triggers.

GRANT { CREATE } TRIGGER
REVOKE { DROP }

Trigger Example

The facing page has an example of creating a simple trigger that places a row into a Salarylog table when the salary is increased by more than 10%.

In this example, the data row before modification is referenced as **oldrow** and the updated data row is referenced as **newrow**.



Trigger Example

```
CREATE SET TABLE Employee
  (Name      CHAR(15),
  Deptid    INTEGER,
  Salary     DECIMAL(10,2),
  Job_Title CHAR(15))
PRIMARY INDEX (Name);
```

```
CREATE SET TABLE Salarylog
  (UserName   CHAR(30),
  EmpName    CHAR(30),
  OldSalary  DECIMAL(10,2),
  NewSalary  DECIMAL(10,2))
PRIMARY INDEX (UserName);
```

Create a trigger that places a new row into the Salarylog table whenever there is a salary increase greater than 10%.

```
CREATE TRIGGER RaiseTrig
AFTER UPDATE OF (Salary) ON Employee
  REFERENCING OLD AS OldRow      NEW AS NewRow
  FOR EACH ROW
    WHEN ((NewRow.Salary - OldRow.Salary)/OldRow.Salary > 0.10)
      ( INSERT INTO SalaryLog
        VALUES (USER, NewRow.Name, OldRow.Salary, NewRow.Salary); );
```

Trigger Example (cont.)

The facing page contains two examples of updating the salary amount to a value greater than 10%. This causes the trigger to fire and place rows into the Salarylog table.



Trigger Example (cont.)

SELECT * FROM Employee;

Name	Deptid	Salary	Job_Title
carol	3333	32000.00	director
debbie	4444	40000.00	president
larry	2222	22000.00	instructor
joe	2222	30000.00	instructor
allan	1111	20000.00	consultant

SELECT * FROM Salarylog;

*** Query completed. 0 row(s) found. 4 field(s) returned.

```
UPDATE Employee SET salary = salary * 1.25 WHERE name = 'allan';
UPDATE Employee SET salary = salary * 1.05 WHERE name = 'debbie';
UPDATE Employee SET salary = 50000 WHERE name = 'carol';
```

SELECT * FROM Employee;

Name	Deptid	Salary	Job_Title
carol	3333	50000.00	director
debbie	4444	42000.00	president
larry	2222	22000.00	instructor
joe	2222	30000.00	instructor
allan	1111	25000.00	consultant

SELECT * FROM Salarylog;

UserName	EmpName	OldSalary	NewSalary
PAYADMIN	allan	20000.00	25000.00
PAYADMIN	carol	32000.00	50000.00

Temporary Table Choices

Generically speaking, there are three types of temporary tables now available with Teradata, any one of which will have advantages over traditional temporary table creation.

Derived tables were incorporated into Teradata under V2R2. Derived tables are always local to a specific query, as they are built with code within the query. The rows of the derived table are stored in spool and discarded when the query finishes. The data dictionary has no knowledge of derived tables.

Volatile Temporary tables (or **Volatile Tables**) are local to a session rather than a specific query, which means that the table may be used repeatedly within a session. Once the session ends, the volatile table is automatically discarded if it has not already been manually discarded. The data dictionary has no knowledge of volatile tables. Space for a volatile table comes from the user's Spool space.

Global Temporary tables (or **Temporary Tables**) are local to a session just as are volatile tables. Unlike volatile tables, global temporary tables are known by the data dictionary where a permanent definition is kept. Global temporary tables are materialized within a session, and then discarded when the session ends. Space for a global temporary table comes from the user's Temporary space.

In this module, we will be looking at the three types of temporary tables, how their implementations differ and when to use each.



Temporary Table Choices

Derived Tables

- Local to the query (table and columns are named within query)
- Incorporated into SQL query syntax (populated in query via SELECT in FROM)
- **Materialized in SPOOL** – Spool rows are discarded when query finishes
- No data dictionary involvement
- Commonly used with aggregation

Volatile Tables

- Local to a session – **uses SPOOL space**
- Uses CREATE VOLATILE TABLE syntax
- Discarded automatically at session end
- No data dictionary involvement

(Global) Temporary Tables

- Local to a session – **uses TEMPORARY space**
- Uses CREATE GLOBAL TEMPORARY TABLE syntax
- Materialized instance of table discarded at session end
- Creates and keeps table definition in data dictionary

Derived Tables Revisited

Derived tables were introduced into Teradata under V2R2. The creation of the derived table is local to the query. A query may have multiple derived tables. These tables may be joined or manipulated much as any other table would be.

The OLAP functions of SQL do not support standard aggregation functions due to their conflicting uses of the GROUP BY clause. This fact makes the OLAP functions excellent candidates for the use of derived tables, in particular when the requirement is to perform a statistical function on an aggregation.

We see in the facing page example that to find the top three selling items across all stores, we must first aggregate the sales by product-id using a derived table. Once we have this aggregation done in spool, we may apply the RANK function to answer the question.

Derived tables are useful, but only exist for the duration of the query. They are not a practical solution if the result is to be used in many follow-on queries. In this case, other types of temporary tables will be more appropriate.



Derived Tables Revisited

Get top three selling items across all stores:

```
SELECT      Prodid, Sumsales, RANK(sumsales) AS "Rank"
FROM        (SELECT prodid, sum(sales) FROM Salestbl GROUP BY 1)
AS tmp (prodid, sumsales)
QUALIFY RANK (sumsales) <= 3;
```

Result:

Prodid	Sumsales	Rank
A	170000.00	1
C	115000.00	2
D	110000.00	3

- Derived table name is “tmp”.
 - The table is required for this query but no others.
 - The query will be run only one time with this data.
- Derived column names are “prodid” and “sumsales”.
- Table is created in spool using the inner SELECT.
- SELECT statement is always in parenthesis following “FROM”.

Volatile Tables

Volatile tables have much in common with derived tables. They are materialized in spool and are unknown to the data dictionary. Unlike derived tables, volatile tables may be used repeatedly throughout a session. They may be dropped at any time manually or automatically at the session end.

Volatile tables require their own CREATE syntax. The table definition is kept in cache and not permanently written to disk. Volatile tables do not survive a system restart.

The **LOG** option indicates the desire for standard transaction logging of “before images” into the transient journal.

The **ON COMMIT DELETE ROWS** option specifies that at the end of a transaction, the table rows should be deleted. While this might seem a bit unusual, it is the default required by the ANSI standard. It may be appropriate in situations where a table is materialized only to produce an aggregation and the table rows are not needed beyond that purpose. (Typically this would occur in a multi-statement transaction.)

The **ON COMMIT PRESERVE ROWS** option provides the more normal situation where the table rows are kept following the end of the transaction.



Volatile Tables

Similar to derived tables:

- Materialized in spool
- No Data Dictionary access or transaction locks
- Table definition kept in cache
- Designed for optimal performance

Different from derived tables:

- Is local to the session, not the query
- Can be used with multiple queries in the session
- Dropped manually anytime or automatically at session end
- Requires CREATE VOLATILE TABLE statement

Example:

```
CREATE VOLATILE TABLE vt_deptsal
, LOG
  (deptno      SMALLINT
   ,avgsal     DEC(9,2)
   ,maxsal     DEC(9,2)
   ,minsal     DEC(9,2)
   ,sumsal     DEC(9,2)
   ,empcnt    SMALLINT)
ON COMMIT PRESERVE ROWS;
```

CREATE Considerations:

- LOG indicates that a transaction journal is maintained.
- NO LOG allows for better performance.
- PRESERVE ROWS indicates keep table rows at TXN end.
- DELETE ROWS indicates delete all table rows at TXN end.
- Volatile tables do not survive a system restart.

Volatile Table Restrictions

Volatile tables must have names that are unique within the user's working database. Even though volatile tables are not known to the data dictionary, if names duplicating dictionary names were allowed, the system would not understand where to locate the requested named object if it could be found in two places.

Up to 1000 volatile tables are allowed on a single session. They must all have unique names. They also must be qualified by the User ID of the session, either explicitly or implicitly. A volatile table cannot belong to a database or a user; it can only belong to a user's session.

While **FALLBACK** is a selectable option, its value is limited for volatile tables. Because they cannot survive a system restart, making a table FALLBACK will not keep a table available following a restart. The only reason to make a volatile table FALLBACK would be to allow creation of the table in the event of a down AMP.

None of the following options are permitted with volatile tables:

- Permanent Journaling
- Referential Integrity
- CHECK constraints
- Column compression
- Column default values
- Column titles
- Named indexes



Volatile Table Restrictions

Restrictions:

- Up to 1000 volatile tables are allowed on a single session.
- Each must have a unique name.
- Volatile tables are always qualified by the session's userid.

Examples:

```
CREATE VOLATILE TABLE username.table1    (Explicit)  
CREATE VOLATILE TABLE table1            (Implicit)  
CREATE VOLATILE TABLE databasename.table1 (Error)
```

Multiple Sessions:

- Each session can use the same VT name (local to session).
- VT name cannot duplicate existing object name for this user.
 - Perm or Temp table names
 - View names
 - Macro names
 - Trigger names

FALLBACK:

Electable but not often useful for VTs. VTs don't survive a system reset.

Options not permitted:

- | | | |
|-------------------------|-------------------------|-----------------|
| • Permanent Journaling | • Referential Integrity | • Named Indexes |
| • CHECK constraints | • Column compression | |
| • Column default values | • Column titles | |

Global Temporary Tables

Global Temporary Tables (also called **Temporary Tables**), unlike volatile and derived tables, have definitions stored in the Data Dictionary. The table itself is materialized by the first SQL DML statement that accesses the table, typically an INSERT SELECT or an INSERT.

Like volatile tables, global temporary tables are local to a session. The materialized instance of the table is not shareable with other sessions. The table instance may be dropped explicitly or it will be automatically dropped at the end of the session. The definition remains in the dictionary for future materialized instances of the table. The base definition may be dropped with an explicit DROP command.

The only privilege required by the user is the DML privilege necessary to materialize the table. Once materialized, no privileges are checked.

A special type of space called “**temporary space**” is used for global temporary tables. Like perm space, temporary space is sustained during a system restart. Global temporary tables are thus able to survive a system restart.

Up to 2000 materialized instances of global temporary tables may exist for a given session.

Two key reasons for the users of global temporary tables are

1. To simplify application code
2. Reduce the a large number of joins for specific tables

Note:

It is common to refer to volatile temporary tables as “**volatile tables**” and refer to global temporary tables simply as “**temporary tables**”.



Global Temporary Tables

Global Temporary Tables:

Are created using CREATE GLOBAL TEMPORARY command.
Require a base definition which is stored in the DD.
Are materialized by first SQL DML statement to access table.

Global Temporary Tables are similar to Volatile Tables:

Each instance of global temp table is local to a session.
Materialized tables are dropped automatically at session end.
Have LOG and ON COMMIT PRESERVE/DELETE options.
Materialized table contents aren't sharable with other sessions.

Global Temporary Tables are different from Volatile Tables:

Base definition is permanent and kept in DD.
Requires DML privileges necessary to materialize the table.
Space is charged against an allocation of "temporary space" - CREATE USER TEMPORARY parameter.
User can materialize up to 2000 global tables per session.
Tables can survive a system restart.

Creating Global Temporary Tables

Temporary tables are created using the CREATE GLOBAL TEMPORARY TABLE command. This stores the base definition of the table in the data dictionary. Like volatile tables, the defaults are to LOG transactions and ON COMMIT DELETE ROWS.

Temporary tables may be altered by the ALTER command to change any attributes of the table, similar to perm tables.

Once the table is accessed by a DML command, such as the INSERT SELECT seen on the facing page, the table is considered materialized and a row is entered into a dictionary view called DBC.Temptables.

Deleting all rows from a temporary table does not de-materialize the table. The instance of the table must be dropped or the session must be ended for the materialized table to be discarded.



Creating Global Temporary Tables

```
CREATE GLOBAL TEMPORARY TABLE gt_deptsal
(deptno      SMALLINT
,avgsal      DEC(9,2)
,maxsal      DEC(9,2)
,minsal      DEC(9,2)
,sumsal      DEC(9,2)
,empcnt      SMALLINT);
```

*Base table definition stored in DD/D
Default is ON COMMIT DELETE ROWS*

```
ALTER TABLE gt_deptsal,
ON COMMIT PRESERVE ROWS;
```

*ALTER TABLE can be done to change
defaults.*

```
INSERT INTO gt_deptsal
SELECT      dept ,AVG(sal) ,MAX(sal) ,MIN(sal)
                  ,SUM(sal) ,COUNT(emp)
FROM        emp
GROUP BY    1;
```

*Table is now materialized.
Row is inserted in DBC.Temptables.*

```
DELETE FROM gt_deptsal;
```

*Table remains materialized until it is
dropped.*

V2R5 – New Features

Teradata Warehouse 7.0 and Teradata Database V2R5 offer features that extend our lead across all dimensions of Data Warehousing. These features are explained in detail in this course:

- Partitioned Primary Index
- Multi Value Compression
- Full Cylinder Read
- Materialized Views
- Roles and Profiles
- Identity Columns
- Query Analysis Tools
- Collect Statistics Enhancements
- Priority Scheduler Enhancements
- TDQM Enhancements
- RSS enhancements
- Updated Limits
- Reconfig
- Conversion and Upgrade

A short description of enhancements and updates to Teradata SQL is included in this course. Students who need to become proficient in using the new Teradata SQL features should take the SQL Web Based Training at Teradata University.



V2R5 – New Features

Single Version of the Business

- Partitioned Primary Indexes
- Cylinder read
- Value List Compression
- 2048 Columns, 64 Column-Indexes
- Identity Column

Trusted, Integrated Environment and Administration

- Index Wizard
- Statistics Wizard and Collection Enhancements
- Database Query Log
- SQL Assistant - WEB Edition
- Availability Enhancements

Data Freshness

- Continuous Update and Manageability
- Faster Join Index Update
- Join Update Performance
- Bulk Update Performance
- Teradata Warehouse Builder Enhancements

Strategic Decision Making

- Analytic extensions: Extended Windows functions and multiple aggregate distincts
- Random Stratified Sampling
- Join Elimination
- Extended Transitive Closure
- Partial Group By
- Early Group By
- Derived Table Rewrite
- Very Large SQL

Tactical and Event Decision Making

- Partial Covering Join Index
- Global Index
- Sparse Index
- Join index Extensions
- ODS Workload Optimization
- Stored Procedure Enhancements

V2R6 – New Features

Teradata Warehouse 8.0 and Teradata Database V2R6 offer features that extend our lead across all dimensions of Data Warehousing.



V2R6 – New Features

Availability

- Reduced Start Time
- Replication Service
- ROLLBACK Performance – rollbacks can use block-at-a-time

Strategic Query Performance

- Improved Random AMP Sampling
- PPI Enhancements
- Increase in “Plan” Cache
- Response Buffer increased to 1 MB
- Table Header Expansion to 128 KB

Applications and Tools Friendliness

- Recursive Query
- Top N Row Operation - returns the first N rows from a query quickly and efficiently, that is, the entire table is not spooled.
- User-Defined Types (UDT) Framework (UDT will be available in future release)

Update Performance

- Iterated Requests – efficient way to execute the same single statement DML operation on multiple data rows
- Improvements in Primary Index operations

Manageability and Administration

- Priority Scheduler Enhancements
- Enhanced Security includes Network Security and Directory Integration

Tactical and Event Processing

- Queue Tables
- IN-List Processing
- LOB Enhancements to Stored Procedures
- External Stored Procedures
- UDF Table Function

Teradata 12.0 – New Features

A list of key new Teradata 12.0 features is shown on the facing page.



Teradata 12.0 – New Features

Performance

- Multi-Level Partitioned Primary Index
- OCES-3 (Optimizer Cost Estimation Sub-system)
- Enhanced Query Rewrite Capability
- Extrapolate Statistics Outside of Range
- Increase Statistics Intervals
- Collect Statistics for Multi-Column NULL Values
- Collect AMP Level Statistics Values
- Parameterized Statement Caching Improvements
- Windowed Aggregate Functions
- Hash Bucket Expansion

Enterprise Fit

- Java Stored Procedures
- Restore/Copy Dictionary Phase
- Restore/Copy to Different Configuration Data
- Phase Performance
- Cursor Positioning for MSR
- UNICODE Support for password control
- Custom Password Dictionary Support
- New Password Encryption Algorithm

Active Enabled

- Online Archive
- Bulk SQL Error Logging Tables
- Full ANSI Merge-Into Capability
- Replication Scalability
- Restartable Scandisk
- Checktable Utility Performance
- Table Functions Without Join-Back

Ease of Use

- Queue Tables
- IN-List Processing
- LOB Enhancements to Stored Procedures
- External Stored Procedures
- UDF Table Function
- TASM: Query Banding, Traffic Cop, Global/Multiple Exceptions, Utility Management, and Open API's
- Enhanced Collection: DBQL & ResUsage
- Enhanced Explain Plan Details
- Stored Procedure Result Sets
- SQL Invocation via External Stored Proc.
- Index Wizard Support for PPI
- Dynamic Result Row Specification on Table Functions
- Normalized AMPUsage View for Coexistence

Cost, Quality, and Supportability

- Compression on Soft/Batch RI Columns
- Dispatcher Fault Isolation

Teradata Limits (Different Releases)

The chart on the facing page highlights the system limits associated with different Teradata releases.



Teradata Limits (Different Releases)

	V2R4.1	V2R5.0/5.1	V2R6.0 – 6.2	12.0
Maximum Vdisk Size	1.26 TB	1.26 TB	1.26 TB	1.26 TB
Maximum Block Size (sectors)	255	255	255	255
Maximum Table Header Size	~64 KB	~64 KB	~128 KB	~128 KB
Number of defined databases/users	4.2 Billion	4.2 Billion	4.2 Billion	4.2 Billion
Concrete Step Size	1 MB	1 MB	1 MB	1 MB
Explain output text max size	No limit	No limit	No Limit	No Limit
Max size character string constant	32 KB	32 KB	32 KB	32 KB
Number of spool tables per query	2048	2048	2048	2048
Number of levels of query nesting (subqueries, nested views, etc.)	64	64	64	64
Maximum # of columns in a Table	256	2048	2048	2048
Maximum # of columns in an Index	16	64	64	64
Column limit for COLLECT STATISTICS	40	512	512	512
Maximum SQL Request Limit Size	64 KB	1 MB	1 MB	1 MB
Maximum SQL Response Buffer Size	64 KB	64 KB	1 MB	1 MB
Maximum # of Hash Bucket Entries	64 KB	64 KB	64 KB	1 MB

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. Which BTEQ setting controls Teradata vs. ANSI mode? _____
2. Which commands will not work in ANSI mode? _____
3. True or False. The SQL Flagger is just a warning device and doesn't affect command execution.
4. True or False. Failure of an individual request in COMMIT mode causes the entire transaction to be rolled back.
5. True or False. Logging off during an explicit transaction without either a COMMIT or ET will always result in a ROLLBACK.
6. True or False. HELP SESSION will show the session mode and the status of the SQL Flagger.
7. Where does a Volatile Temporary table get its space from? _____
8. Where does a Global Temporary table get its space from? _____
9. A trigger executes (fires) when either an _____, _____, or _____ statement modifies a specified column or columns in a table.

Teradata Training

Notes

Module 32



Introduction to Application Utilities

After completing this module, you should be able to:

- Identify the Application Utilities.
- Describe how the Application Utilities interface with the Teradata Database.
- State the advantage of using a utility over other access methods.
- Match Teradata Parallel Transporter operators with the corresponding Teradata utility.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Application Utilities	32-4
Application Utilities Environments	32-6
Application Utilities on a Local Area Network	32-6
Application Utilities on a Mainframe Host.....	32-6
Application Development	32-8
Utility advantages	32-8
Transferring Large Amounts of Data.....	32-10
INSERT/SELECT: The Fast Path.....	32-12
Multi-Statement Insert/Select Example	32-14
DELETE (ALL): The Fast Path	32-16
Using an INMOD Routine	32-18
Teradata Parallel Transporter.....	32-20
Teradata Parallel Transporter Operators	32-22
Referential Integrity and Load Utility Issues	32-24
Application Utility Checklist	32-26
Application Utility Summary	32-28
Review Questions	32-30

Application Utilities

The Teradata database provides several Application Utilities for processing large numbers of INSERTs, UPDATEs, and DELETEs in a batch environment.

Each utility exploits the capabilities provided by the Teradata parallel architecture for a specific data maintenance or batch-processing activity.

Teradata application utilities are supported on several hardware platforms including a wide range of channel-connected mainframes.

Regardless of the host platform, however, all access between the host and the Teradata database relies on the Call Level Interface (CLI), a series of callable subroutines that reside in the host's address space.

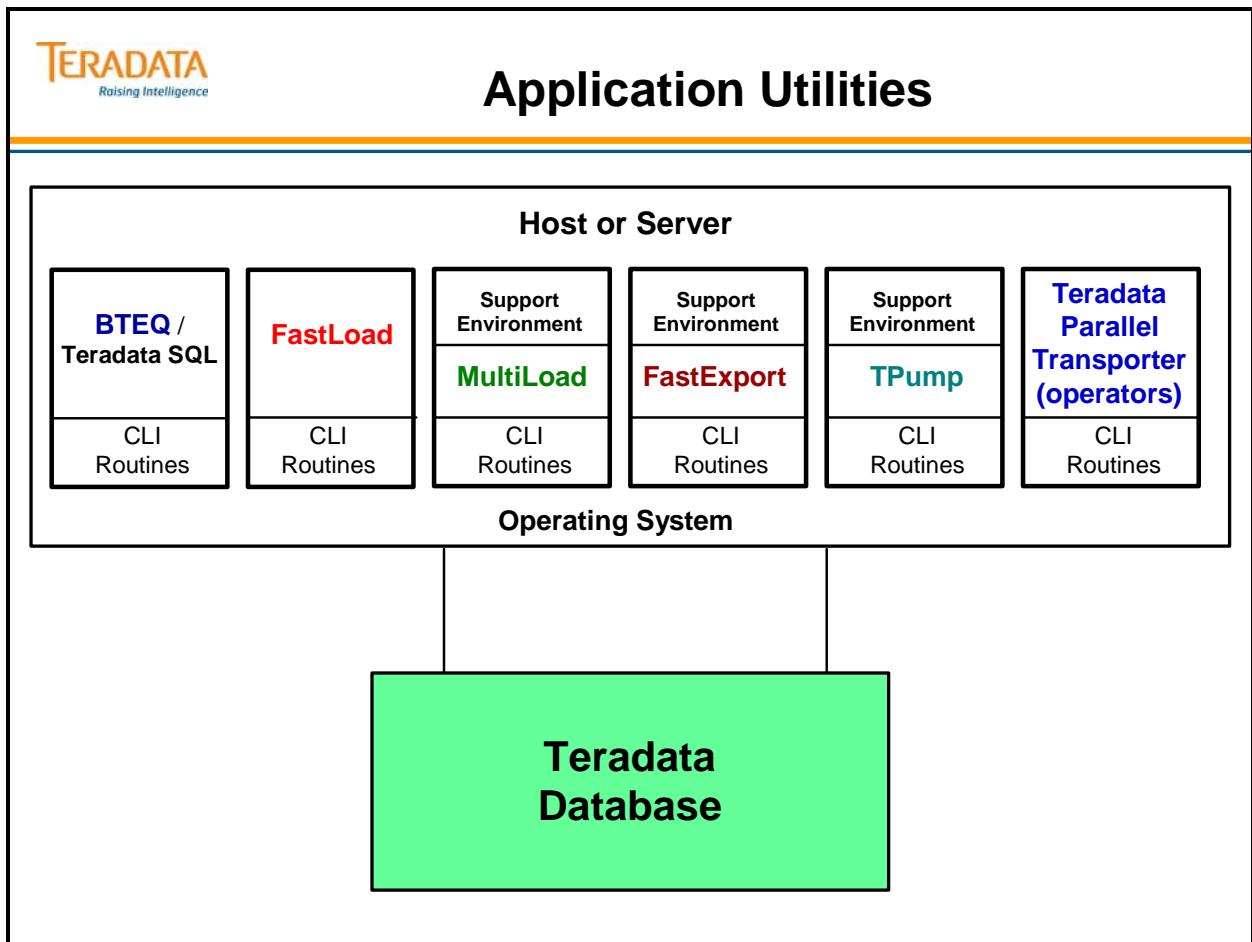
CLI is responsible for creating and managing the parcels that travel back and forth between Teradata and the host. It permits the host to send multiple tasks (sessions) to Teradata at the same time.

CLI is the vehicle that makes parallel access possible.

BulkLoad

The BulkLoad utility (not shown on facing page) is an older utility that executed on a channel-attached mainframe. BulkLoad was one of the original Teradata loader utilities and has been replaced by the more efficient utility, TPump. BulkLoad supported SELECT, INSERT, UPDATE, DELETE and submits SQL transactions at SQL speed.

In addition to the main processor being used on the host platform, most activities use special protocols on the database engine itself.



Application Utilities Environments

Even though a single UNIX node employs an innovative architecture that uses the combined power of multiple tightly-coupled processors as a powerful UNIX mainframe, it is preferred that Application Utilities execute on a separate server, a different SMP node, or on a mainframe host. This server may be another node within the configuration or a separate server that is LAN connected. If the node is part of the configuration, then it communicates directly over the BYNET, thus avoiding performance constraints associated with a channel connection.

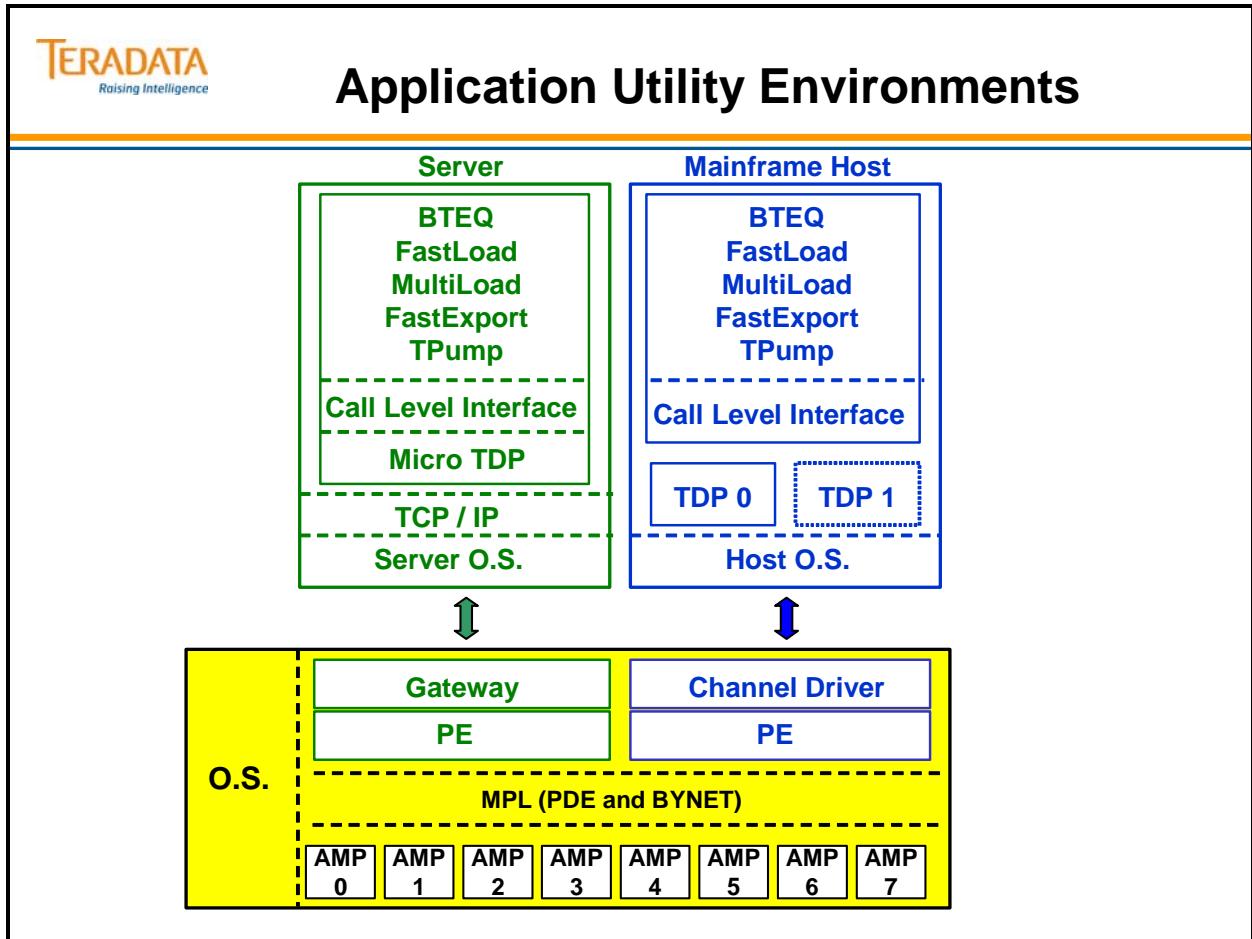
Application Utilities on a Local Area Network

Each PC or workstation accessing the Teradata database over a LAN has a single-threaded version of TDP (Teradata Director Program), which is known as Micro TDP (MTDP).

Although the capacities of PCs and workstations are increasing rapidly, performance can be constrained by memory and disk limitations. For this reason PCs are less likely to handle the large amounts of data normally associated with the Application Utilities.

Application Utilities on a Mainframe Host

Application Utilities are frequently executed on mainframes hosts using the mainframe channel connections to load/unload data to/from Teradata. The TDP provides session control for the mainframe host.



Application Development

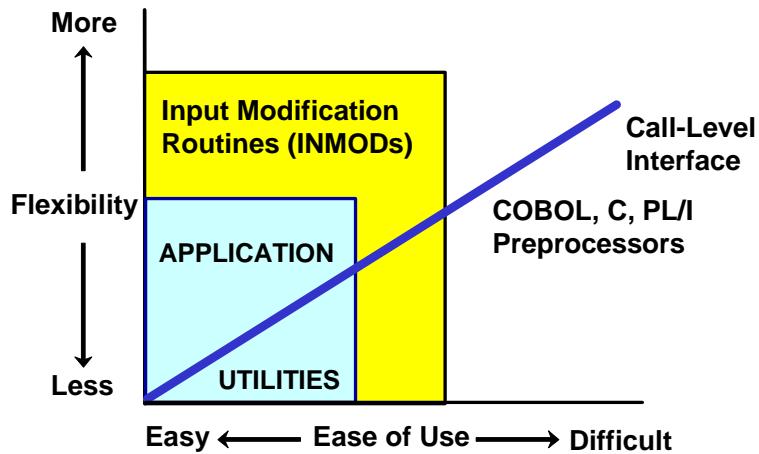
All access to the Teradata database is accomplished using the Call Level Interface. All the Teradata utilities are programs written in a 2nd or 3rd generation language. Depending upon the programmer's skill, programs written using the CLI are extremely flexible and can accomplish any function supported by the Teradata database. These types of programs are complex and difficult both to write and maintain.

Less complex, but still requiring a high level of programming ability, SQL statements can be imbedded in COBOL, PL/I or C Programs using the Teradata Preprocessor. Application design, coding and implementation are lengthy processes.

Utility advantages

The Teradata utilities represent an alternative to custom development. They are easy to use, simple to maintain, and quick to implement. Moreover, they have all been tested, documented, and are vendor-supported.

Application Development



Selection of the right vehicle can be crucial to the success of the application:

- How difficult is it to implement?
- How difficult is it to maintain?

Use the application utilities wherever possible:

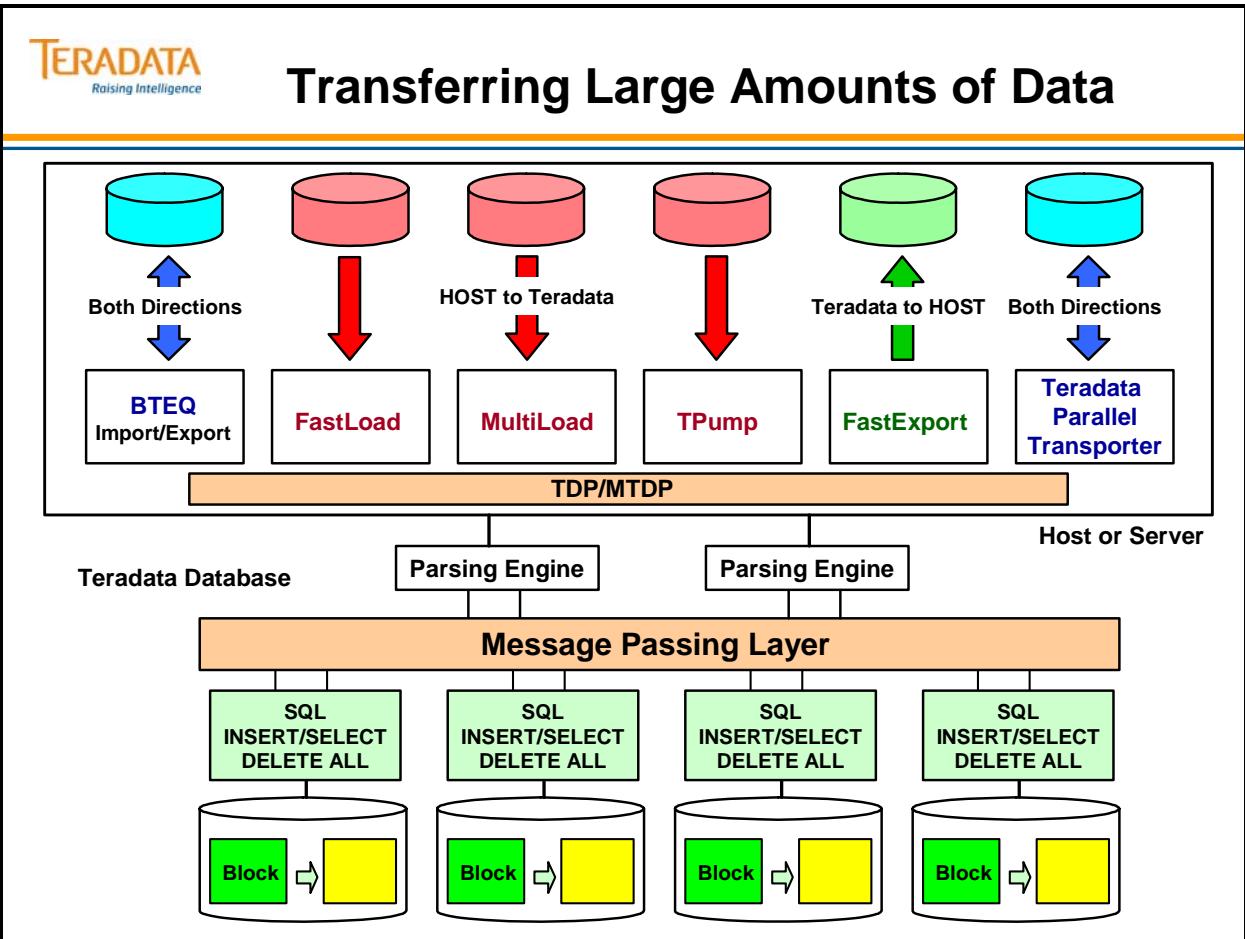
- Offer the least complexity.
- Take full advantage of parallel processing.

Transferring Large Amounts of Data

The Teradata application utilities reside in the host computer, whether it be the Application Processor, a mainframe, or a workstation.

- **BTEQ** supports all 4 DMLs: SELECT, INSERT, UPDATE and DELETE. BTEQ also supports IMPORT/EXPORT protocols.
- **FastLoad**, **MultiLoad**, and **TPump** transfer data from the host to Teradata.
- **FastExport** performs high volume SELECTs to export data from Teradata to the host.

Apart from the application utilities themselves, there is a special optimization of the **SQL INSERT/SELECT** and **DELETE** that deserve some attention.



INSERT/SELECT: The Fast Path

The Teradata INSERT/SELECT is optimized to populate one table in the Teradata database from another at high speed provided two conditions are true:

1. The tables must have the same Primary Index (PI) **and**
2. The target table must be empty.

In almost all computer systems, the disk is the slowest component and therefore defines a performance bottleneck that limits throughput. While you cannot reasonably avoid writing I/Os to disk, you can certainly try to keep them to a minimum by writing rows to disk *a block at a time*.

Teradata stores rows in data blocks sorted ascending by *row hash*. Teradata never mixes rows of different tables in the same block, and rows never span blocks.

If both the source table and the target table have the same Primary Index, they will be on the same AMP and already hashed, formatted, and sorted in data blocks. During this kind of INSERT/SELECT operation each AMP can locally assemble the blocks for the new table in memory, and, providing the target table is empty, write entire blocks to disk with a single I/O.

Multiple INSERT/SELECT operation

In addition to the above, BTEQ permits you to populate a single initially empty table from multiple source tables at high speed. To do this, use a multiple statement INSERT/SELECT. Again, *all* tables must have the same Primary Index.

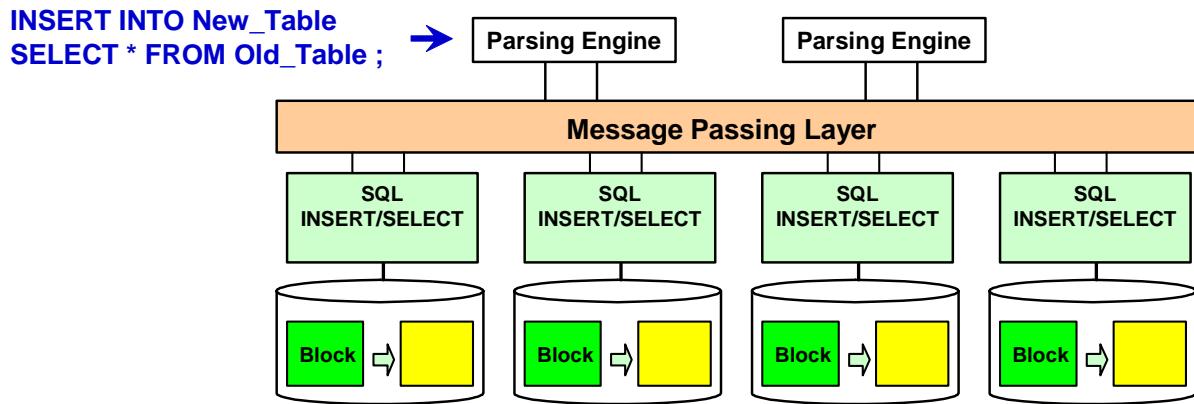
Example:

```
INSERT into T1 SELECT * FROM T2
;INSERT INTO T1 SELECT * FROM T3
;INSERT INTO T1 SELECT * FROM T4;
```

Each AMP selects the (presorted) rows from all source tables, builds the data blocks in memory, and writes the new table a block at a time.



INSERT/SELECT: The Fast Path



INSERT / SELECT achieves highest performance if:

- Target table is empty, **AND**
- Source and target tables have same Primary Index.

Advantages of using optimized INSERT / SELECT:

- One WRITE to the Transient Journal – instantaneous rollback for aborted Insert>Select statements.
- Data copied and written to disk a block at a time.
- No data redistribution over the BYNET.

Multi-Statement Insert/Select Example

Look at the example on the facing page. Using multiple Regional Sales History tables, a single summary table is built by combining summaries from the different regions.

A summarization may be done for each region. Summarizations then are inserted into a single table using a multi-statement Insert Select statement.

All multi-statement Insert Select statements output to the same spool table. The output is sorted and inserted into an empty table.

A multi-statement request is formed by semicolon placement in BTEQ, as shown on the facing page, or by placing statements in a single macro.

If the statements were executed separately, only the first statement is inserted into an empty table.



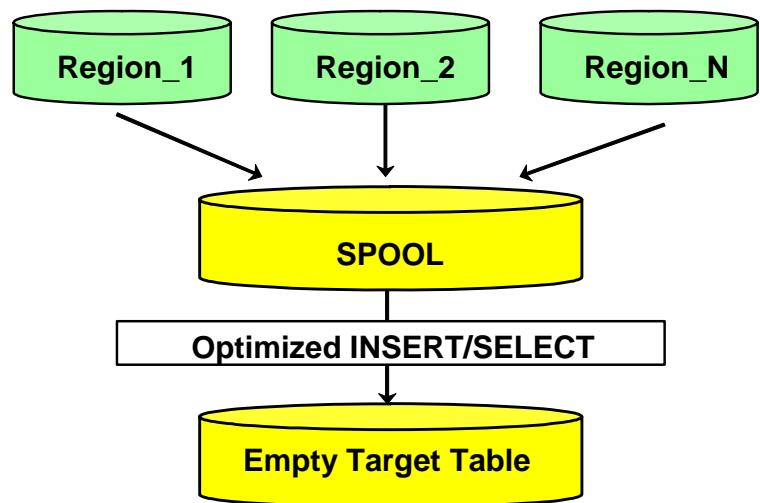
Multi-Statement INSERT/SELECT Example

```

INSERT INTO
  SELECT
    Summary_Table
    store, region,
    SUM(sales),
    COUNT(sale_item)
  FROM
    Region_1
    GROUP BY
      1, 2

; INSERT INTO
  SELECT
    Summary_Table
    store, region,
    SUM(sales),
    COUNT(sale_item)
  FROM
    Region_2
    GROUP BY
      1, 2
    ...

; INSERT INTO
  SELECT
    Summary_Table
    store, region,
    SUM(sales),
    COUNT(sale_item)
  FROM
    Region_N
    GROUP BY
      1, 2
;
  
```

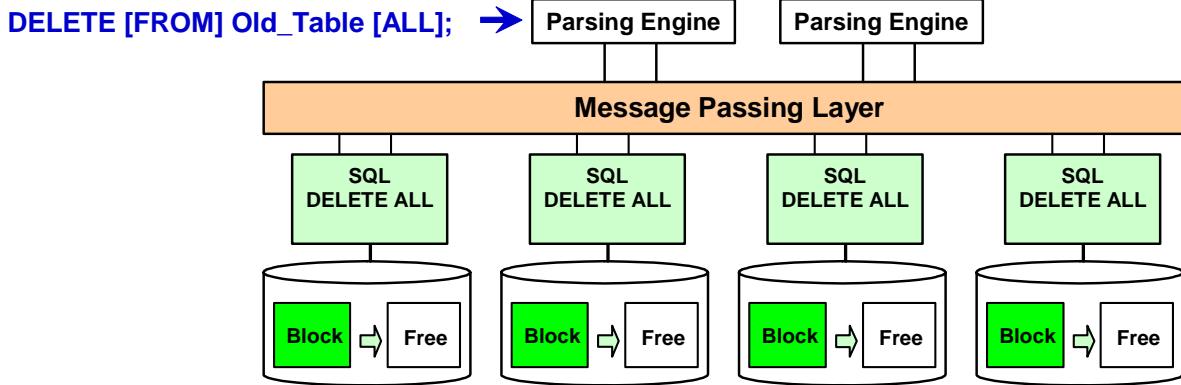


DELETE (ALL): The Fast Path

Like INSERT/SELECT, you can achieve high performance using DELETE (ALL).



DELETE (ALL): The Fast Path



DELETE achieves its highest performance when deleting all of the rows in a table.

High performance is achievable because:

- Transient Journal is not used to store before-images of deleted rows.
- DELETEs are done at the cylinder index / master index level.

ANSI Transaction Mode: In ANSI Transaction mode, to achieve the DELETE Fast Path performance, the COMMIT needs to be included as part of a multi-statement request.

```
DELETE Old_Table
; COMMIT ;
```

Using an INMOD Routine

The application utilities allow input data to be read or pre-processed by a user-written INMOD routine. The routines call the defined program module, which is responsible for delivering an input record.

An INMOD is a user exit routine used by utilities to supply or preprocess input records.

Major functions performed by an INMOD include:

- Generating records to be passed to the utility.
- Validating a data record before passing it to the utility.
- Reading data directly from one or more database systems like IMS, Total.
- Converting fields in a data record before passing it to the utility.

As we discuss each utility, we will identify the return codes used.

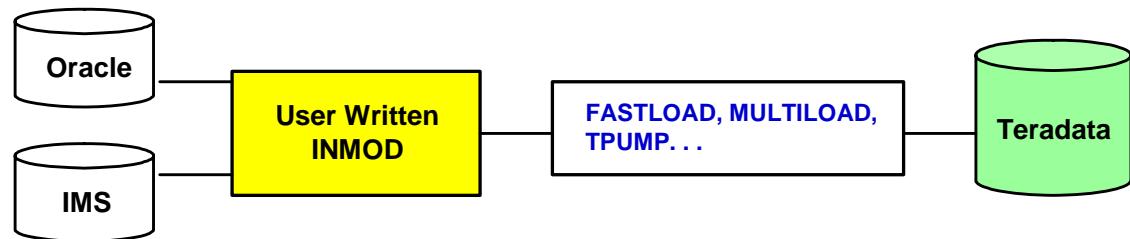


Using an INMOD Routine

An **INMOD** is a user-written program that allows for pre-processing of the data before giving the data to the utility – possibly acting as a data filter to the utility.

An INMOD routine can perform various functions:

- Validate a data record.
- Read data directly from one or more database systems, allowing the creation of a composite input data record, and avoiding the need for an intermediate tape or disk.
- Select specific records for input to the Teradata Database.
- Perform data conversions not supported by the application utilities.
- Add or change data fields in the records.



An **OUTMOD** is a user-written program that does post-processing of the data after the utility (e.g., FastExport) has retrieved the data.

Teradata Parallel Transporter

Teradata Parallel Transporter is the replacement for Teradata Warehouse Builder (TWB). This utility is effectively an object-oriented software system that executes multiple instances of data extraction, transformation, and load functions in a scalable, high-speed parallel processing environment.

Teradata Parallel Transporter is scalable and enables end-to-end parallelism. The previous versions of utilities (like FastLoad) allow you to load data into Teradata in parallel, but you still have a single input stream. Teradata Parallel Transporter allows you to run multiple instances of the extract, optional transformation, and load. You can have as many loads as you have sources in the same job. With multiple sources of data coming from multiple platforms, integration is important in a parallel environment.

Teradata Parallel Transporter eliminates the need for persistent storage. It stores data into data buffers so you no longer need to write data into a flat file. Since you don't need flat files, you no longer need to worry about a 2 GB file limit.

Teradata Parallel Transporter provides a single scripting language. You can do the extract, some transformation, and loads all in one SQL-like scripting language.

Once the dynamics of the language are learned, you can perform multiple tasks with a single script.

Teradata Parallel Transporter supports FastLoad INMODs, FastExport OUTMODs, and Access Modules to provide access to all the data sources you use today.

Teradata Parallel Transporter also provides a Direct API interface that can be used by Third Party partners. They can effectively write 'C' code to directly load/unload data to/from Teradata.



Teradata Parallel Transporter

Teradata Parallel Transporter (replacement for Teradata Warehouse Builder – TWB) can load data into and export data from the Teradata database.

It is effectively **a parallel load and export utility**. Characteristics of this utility include:

High performance

- Parallel Export and Load operators eliminate sequential bottlenecks.
- Data Streams eliminate the overhead of intermediate (persistent) storage.
- Scalable
- End-to-end parallelism

Easy to use

- Single scripting language
- Access to various data sources

Extensible

- Direct API enables Third Party partners to write 'C' code to directly load/unload data to/from Teradata.

Teradata Parallel Transporter Operators

Operators are the software components that provide the actual data extraction, transformation, and load functions in support of various data stores.

This utility supports different types of operators, where the operator type signifies the primary function of the operator:

- Producer Data extraction functions (e.g., Export operator):
 - Get data from the Teradata Database or from an external data store
 - Generate data internally
 - Pass data to other operators by way of the operator interface
- Consumer Data loading functions (e.g., Load operator):
 - Accept data from other operators by way of the operator interface
 - Load data into the Teradata Database or to an external data store
- Filter Data transformation functions such as:
 - Selection, validation, cleansing, and condensing

General notes about this utility and its operators.

- The FastLoad INMOD and FastExport OUTMOD operators support the current FastLoad and FastExport INMOD/OUTMOD features.
- The Data Connector operator is an adapter for the Access Module or non-Teradata files.
- The SQL Select and Insert operators submit the Teradata SELECT and INSERT commands.
- The Load, Update, Export and Stream operators are similar to the current FastLoad, MultiLoad, FastExport and TPump utilities, but built for the TWB parallel environment.

The INMOD and OUTMOD adapters, Data Connector operator, and the SQL Select/Insert operators are included when you purchase the Infrastructure. The Load, Update, Export and Stream operators are purchased separately.

To simplify these new concepts, the facing page compares the Teradata Parallel Transporter Operators with the classic utilities.



Teradata Parallel Transporter Operators

TPT Operator	Teradata Utility	Description
LOAD	FastLoad	A consumer-type operator that uses the Teradata FastLoad protocol. Supports Error limits and Checkpoint/ Restart. Both support Multi-Value Compression and PPI.
UPDATE	MultiLoad	Utilizes the Teradata MultiLoad protocol to enable job based table updates. This allows highly scalable and parallel inserts and updates to a pre-existing table.
EXPORT	FastExport	A producer operator that emulates the FastExport utility.
STREAM	TPump	Uses multiple sessions to perform DML transactions in near real-time.
DataConnector	N/A	This operator emulates the Data Connector API. Reads external data files, writes data to external data files, reads an unspecified number of data files.
ODBC	N/A	Reads data from an ODBC Provider.

Referential Integrity and Load Utility Issues

FastLoad and MultiLoad cannot be used to load data into tables that have standard or batch referential integrity constraints defined. FastLoad and MultiLoad can be used to load data into tables that have soft referential integrity constraints (REFERENCES WITH NO CHECK) defined.

First approach (probably easier in many situations):

1. Create the tables and define the Primary Keys. Primary Keys (referenced columns) must be NOT NULL and will be implemented as unique index (either primary or secondary).
2. Populate the tables.
3. Create the Foreign Key references.

If any row in the referencing column violates the RI constraint, the Reference constraint is created and an error table (tablename_0) is automatically created.

Second approach:

1. Create the tables and define the Primary Keys. Primary Keys (referenced columns) must be NOT NULL and will be implemented as unique index (either primary or secondary).
2. Create the Foreign Key references.
3. Populate the tables.

If you are populating the tables with INSERT/SELECT and using the second approach, when a foreign key violation is encountered, the INSERT/SELECT fails and the entire INSERT/SELECT is rolled back.



Referential Integrity and Load Utility Issues

Tables that have **Standard** or **Batch** reference constraints **cannot** be loaded with FastLoad, MultiLoad, or with TPT LOAD/UPDATE operators.

Standard RI – REFERENCES

Batch RI – REFERENCES WITH CHECK OPTION

Soft RI – REFERENCES WITH NO CHECK OPTION (can be loaded with FastLoad, MultiLoad, or Teradata Parallel Transporter (TPT) LOAD/UPDATE operators)

There are two different approaches to establishing RI and populating tables.

First Approach (recommended):

1. Create the tables and define the Primary Keys.
2. Populate the tables.
3. Create the Foreign Key references.

Second approach:

1. Create the tables and define the Primary Keys.
2. Create the Foreign Key references.
3. Populate the tables with SQL or TPump.

Application Utility Checklist

We will complete the checklist on the opposite page as we discuss each utility. It will help you to evaluate capabilities and advantages.

As we discuss each one, it will become apparent that they have been developed over time to address evolving user needs.

Although not strictly a utility, BTEQ can be considered the grandparent of them all. BTEQ was initially developed as a means of sending the SQL to the Teradata database without having to write a complex program using the CLI for each and every query.



Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions					
DML Functions					
Multiple DML					
Multiple Tables					
Multiple Sessions					
Protocol Used					
Conditional Expressions					
Arithmetic Calculations					
Data Conversion					
Error Tables					
Error Limits					
User-written Routines					
Uses Support Environment					

Application Utility Summary

The facing page summarizes some of the key points of this module.



Application Utility Summary

- **BTEQ** supports **SELECT, INSERT, UPDATE, DELETE**.
 - BTEQ **INSERT/SELECT** and **DELETE (ALL)** can provide a fast effective method to perform some tasks.
- **FastLoad, MultiLoad, and TPump** transfer data from the host to Teradata.
- **FastExport** transfers data from Teradata to the host.
- Utilities offer the least complex solutions for an application, and can take advantage of parallel processing.
 - Utilities permit the use of INMODs and/or OUTMODs for pre- or post-processing data.
 - There is often more than one way to set up your application, but there may be one that is fastest or most effective.
- **Teradata Parallel Transporter** can load data into and export data from any accessible database object in the Teradata Database or other data store for which there exists an access operator.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Answer True or False.

1. **True or False.** With MultiLoad, you can import and export data.
2. **True or False.** In Teradata mode, a BTEQ DELETE ALL function does not use the Transient Journal to store before-images of deleted rows.
3. **True or False.** An INSERT/SELECT of 1,000,000 rows into an empty table is only slightly faster than an INSERT/SELECT of 1,000,000 rows into a table with 1 row.

Match the Teradata Parallel Transporter operator with the corresponding Teradata utility.

- | | |
|------------------------------------|---------------|
| 1. <input type="checkbox"/> UPDATE | A. MultiLoad |
| 2. <input type="checkbox"/> STREAM | B. FastLoad |
| 3. <input type="checkbox"/> LOAD | C. FastExport |
| 4. <input type="checkbox"/> EXPORT | D. TPump |

Teradata Training

Notes

Module 33



BTEQ

After completing this module, you will be able to:

- Use .EXPORT to SELECT data from the Teradata database to another computer.
- State the purpose of the four types of BTEQ EXPORT.
- Use .IMPORT to process input from a host-resident data file.
- Use Indicator Variables to preserve NULLs.
- Describe multiple sessions, and how they make parallel access possible.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

BTEQ	33-4
Using BTEQ Conditional Logic	33-6
BTEQ Error Handling	33-8
BTEQ EXPORT – Example 1	33-10
4 Types of BTEQ .EXPORT	33-12
1. Field Mode (REPORT)	33-12
2. Record Mode (DATA)	33-12
3. INDICDATA	33-12
4. Data Interchange Format (DIF).....	33-12
LIMIT.....	33-12
BTEQ Data Modes	33-14
BTEQ EXPORT – Example 2	33-16
UNIX Variables in a BTEQ script	33-16
BTEQ EXPORT – Example 3	33-18
Indicator Variables	33-20
Determining the Logical Record Length with Fixed Length Columns.....	33-22
Determining the Logical Record Length with Variable Length Columns	33-24
Determining the Logical Record Length with .EXPORT INDICDATA.....	33-26
.IMPORT (for Network-Attached Systems)	33-28
VARTEXT Notes.....	33-28
AXSMOD Example	33-28
.IMPORT (for Channel-Attached Systems).....	33-30
.PACK	33-32
.REPEAT.....	33-32
BTEQ IMPORT – Example 1	33-34
BTEQ IMPORT – Example 2	33-36
BTEQ IMPORT – Example 3	33-38
Multiple Sessions	33-40
.SET SESSIONS	33-42
Parallel Processing Using Multiple Sessions to Access Individual Rows	33-44
When Do Multiple Sessions Make Sense?.....	33-46
Application Utility Checklist	33-48
Review Questions	33-50
Lab Exercise 33-1	33-52
Lab Exercise 33-2	33-56

BTEQ

BTEQ (Batch/Basic Teradata Query Language) was originally designed as a means to send SQL requests from a host file to the Teradata database and deliver the response in the format required.

BTEQ can be used for both exporting and importing data.

BTEQ is available on every platform supported by Teradata and is widely used. Even though BTEQ is frequently used in a pseudo-interactive mode, this course concentrates on those features that render it a useful vehicle for batch or data maintenance operations.

A BTEQ session provides a quick and easy way to access a Teradata Database. In a BTEQ session, you can do the following:

- Enter Teradata SQL statements to view, add, modify, and delete data.
- Enter BTEQ commands.
- Enter operating system commands.
- Create and use Teradata stored procedures.



BTEQ

- General purpose command-based utility for submitting SQL requests to the Teradata database.
 - BTEQ (Basic Teradata Query) operates in either a Batch or Interactive mode.
 - BTEQ is a CLI-based utility.
- Runs on every supported platform — laptop to mainframe.
- Exports data to a client system from the Teradata database:
 - As displayable characters suitable for reports, or
 - In native host format, suitable for other applications.
- Imports data from a host or server data file and can use that data within SQL statements (INSERT, UPDATE, or DELETE).
- Flexible and easy-to-use report writer.
- Limited ability to branch forward to a LABEL, based on a return code or an activity count.
- BTEQ does error reporting, not error capture.
- The .OS command allows the execution of operating system commands.

Using BTEQ Conditional Logic

A feature of BTEQ that can be effectively used to improve application performance is its ability to branch forward in a script based on a test of either an error code or an activity count. While this is not a true loop function, it can be used to avoid unnecessary, time-consuming steps.

In the example on the facing page, the script is designed to delete a table. If the delete is successful, the return code is zero, and the system knows that the table already exists. It does not need to create it.

The example script also tests the number of rows that qualify for insertion into the table. Based on the result of the test, alternative subsequent actions can be performed.



Using BTEQ Conditional Logic

The Bank offers a number of special services to its Million-Dollar customers.

```

DELETE FROM Million_Dollar_Customer ALL;
.IF ERRORCODE = 0 THEN .GOTO TableOK
CREATE TABLE Million_Dollar_Customer
(Account_Number           INTEGER
,Customer_Last_Name      VARCHAR(20)
,Customer_First_Name     VARCHAR(15)
,Balance_Current         DECIMAL(9,2));
.LABEL TableOK
INSERT INTO Million_Dollar_Customer
SELECT   A.Account_Number, C.Last_Name, C.First_Name, A.Balance_Current
FROM     Accounts A      INNER JOIN
          Account_Customer AC  INNER JOIN
          Customer C
ON       C.Customer_Number = AC.Customer_Number
ON       A.Account_Number = AC.Account_Number
WHERE    A.Balance_Current GT 1000000;
.IF ACTIVITYCOUNT > 0 THEN .GOTO Continue
.QUIT
.LABEL Continue

```

DELETE all rows from the `Million_Dollar_Customer` table.

IF this results in an error (non-zero), **THEN** create the table, **ELSE** attempt to populate using `INSERT/SELECT`.

IF some rows are inserted (`ACTIVITYCOUNT>0`) **THEN** arrange services, **ELSE** terminate the job.

BTEQ Error Handling

The BTEQ error handling capability permits you to assign various severity values to specific types of errors. Use these values to make a decision to take a specific action based on the occurrence of either a specific type of error or a high-watermark value.



BTEQ Error Handling

```
.SET ERRORLEVEL 2168          SEVERITY 4,  
                  (2173, 3342, 5262) SEVERITY 8  
.SET ERRORLEVEL UNKNOWN      SEVERITY 16  
SELECT      .....  
FROM       ..... ;  
.IF ERRORLEVEL >= 16 THEN .QUIT 16 ;
```

You can assign an error level (SEVERITY) for each error code returned and make decisions based on the level you assign.

ERRORCODE → Tests last SQL statement only.

ERRORLEVEL → Set by user and retained until reset.

Capabilities:

- Customize mapping from error code to ERRORLEVEL.
- .SET MAXERROR <integer> defines termination threshold.

BTEQ EXPORT – Example 1

BTEQ and SQL commands are frequently maintained in the same file or script and may be submitted either in-stream or with a .RUN command.

BTEQ typically delivers a default response to all SQL queries that includes a helpful message along with diagnostic information about the elapsed time taken to perform the query. In its pseudo-interactive environment, this information is captured in the single default output file. This mixed output typically renders the data unsuitable for some purposes.

The .EXPORT feature of BTEQ, which names an output file, provides the ability to separate the report or output data from the accounting information.

The only difference between BTEQ .EXPORT to a LAN or UNIX-based environment (such as a WorldMark system) and a mainframe host is in the way the output file name is specified.

When identifying the destination for the output file, for BTEQ running on an IBM host, use the parameter, “DDNAME =”. For LAN and network servers, use the expression “FILE =”.

Note that BTEQ statements are distinct from SQL statements; they begin with a period (.) and do not require a trailing semi-colon. (The trailing semi-colon is required for other application utilities.)



BTEQ EXPORT – Example 1

export1.btq

BTEQ Script

```
.LOGON tdp1/user1,passwd1
.OS rm /home/user1/datafile1
.EXPORT DATA FILE=/home/user1/datafile1
  SELECT Account_Number
  FROM Accounts
  WHERE Balance_Current LT 100;
.EXPORT RESET
.QUIT
```

Note: BTEQ will append data to an existing file. To avoid this, use the .OS "remove" command to ensure that the file doesn't exist.

bteq < export1.btq



Default Output

Logon complete
1200 Rows returned
Time was 15.25 seconds

datafile1

12348009
19450824
23498763
23748091
85673542
19530824
92234590
:

Data file of
Account
Numbers

4 Types of BTEQ .EXPORT

1. Field Mode (REPORT)

When submitting BTEQ requests to a Teradata database, you may have noted that output is always provided with column headings and underscores, with numerics aligned to the right, characters to the left, and all output displayed in the center of the screen or report. This is Field mode, the default output of BTEQ (suitable for reports).

REPORT – output is truncated to 254 characters

REPORTWIDE – output is truncated to 32 KB (only supported in some releases)

The REPORT format contains an option (not shown) of NOBOM or BOM (Byte Order Mark). This option identifies if the BOM is to be added or not when exporting Unicode data. This is associated with Unicode UTF text formatting. The default is to add the BOM.

2. Record Mode (DATA)

You might require output data in a flat-file format with binary data, no headings, etc. Request output in this format by using Data mode.

3. INDICDATA

Host computer systems rarely have the built-in capability to recognize or handle NULL data. You might need to use INDICDATA if the data contains NULL columns.

4. Data Interchange Format (DIF)

Use the DIF output option if you need data in a format suitable for PC-based applications such as Lotus 1-2-3 and Microsoft Excel. The DATALABELS option includes the column titles of the selection results as the first row in the DIF file.

The DIF format also contains an option (not shown) of NOBOM or BOM (Byte Order Mark). This option identifies if the BOM is to be added or not when exporting Unicode data. This is associated with Unicode UTF text formatting. The default is to add the BOM.

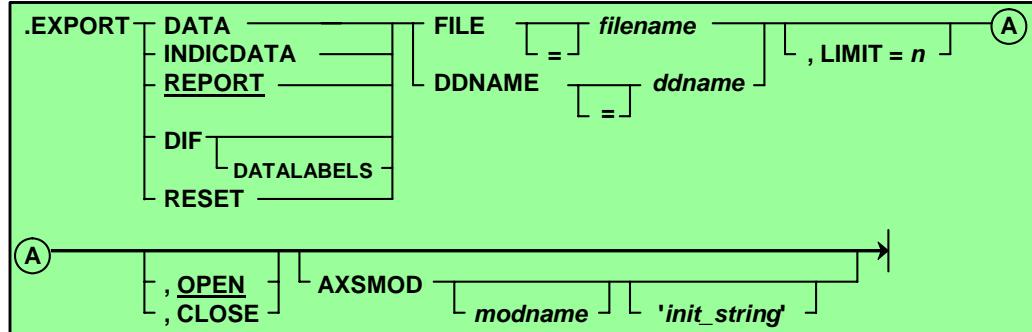
LIMIT

The LIMIT feature of the EXPORT command is useful to application programmers who require a small subset of pre-existing data to test applications. Remember, however, that BTEQ is a host-resident utility, and that BTEQ commands are not seen by the Teradata database. For this reason, while the LIMIT function is observed by BTEQ, it is not seen by the Teradata database parser.

Note: The parameters must be on a single line. The LIMIT parameter must be on the same line as the .EXPORT.



BTEQ .EXPORT



.EXPORT DATA	Sends results to a host file in record mode.
.EXPORT INDICDATA	Sends query results that contain indicator variables to a host file. Allows Host programs to deal with nulls.
.EXPORT REPORT	Sends results to a host file in field mode. Data set contains column headings and formatted data. Data is truncated if exceeds 254 (REPORT).
.EXPORT DIF	Output converted to Data Interchange Format – used to transport data to various PC programs.
.EXPORT RESET	Reverses the effect of a previous .EXPORT and closes the output file.
LIMIT n	Sets a limit on number of rows captured.
OPEN/CLOSE	Output Data Set or File is either OPEN or CLOSEd during RETRY
AXSMOD	Access module used to export to tape

BTEQ Data Modes

The terms “FIELD” and “RECORD” as mode-names for BTEQ may not be apparent until you consider the way parcels are sent by the Call Level Interface back and forth to the Teradata database.

If the application needs response data as formatted, displayable characters suitable for reports, specify FIELD Mode (the default) with an .EXPORT REPORT command. Field mode instructs the Teradata database to return formatted data parcels with a series of header parcels providing details of column headings, data types, and lengths. BTEQ then formats the responses prior to delivering them to the user. Each returning parcel contains a single FIELD of information.

If you require binary data for further activity, use .EXPORT DATA to provide a flat-file response. Each returning parcel will contain a complete RECORD.

INDICDATA mode is needed because host computer operating systems have no way of representing missing or unknown data (NULLs), but this functionality is required for Relational Database Systems.

AMPs provide output data consistent with the expectations of the particular type of host and convert output for fixed-length columns containing NULLs to zeros or spaces. Because zero is a number and a space is a valid character value, NULLs can be misunderstood by any application required to re-process the data. A flag is needed to indicate that, despite the values contained in this field, it should be treated as NULL.

Use INDICDATA mode to precede the output data record with the number of bytes representing individual bit settings for each field returned to the host. Teradata application utilities can then be instructed to observe this convention and thereby ensure complete integrity of the data.



BTEQ Data Modes

Field mode is set by : .EXPORT REPORT

<u>Column A</u>	<u>Column B</u>	<u>Column C</u>
1	2	3
4	5	6
7	8	9

Transfers data one column at a time with numeric data converted to character.

Record mode is set by : .EXPORT DATA

f1	f2	f3
f1	f2	f3
f1	f2	f3

Transfers data one row at a time in host format. Nulls are represented as zeros or spaces.

Indicator mode is set by: .EXPORT INDICDATA

Indic. Byte(s)	f1	f2	f3
Indic. Byte(s)	f1	f2	f3
Indic. Byte(s)	f1	f2	f3

Transfers data one row at a time in host format, sending an indicator variable for nulls. Nulls are represented as zeros or spaces.

BTEQ EXPORT – Example 2

The facing page displays a simple BTEQ .EXPORT script which limits the size of the output to 100 rows. Note the LIMIT parameter on the same line as the .EXPORT statement.

The **tee** command sends standard output (stdout) to the display device and to a specified file.

UNIX Variables in a BTEQ script

The following technique can be used to support variables in a UNIX script:

Example:

```
cat cr_bteq.sh

echo please enter the directory where all your files reside:
read in_dir

bteq << !
.run file = ${in_dir}/logon.btq;
.export data file = ${in_dir}/output_data.txt, limit=100
select * from au2.trans;
.export reset;
!
```



BTEQ EXPORT – Example 2

This example illustrates the use of the LIMIT parameter to reduce the amount of exported data.

export2.btq

```
.LOGON tdp1/user1,passwd1
.OS rm datafile2
.EXPORT DATA FILE = datafile2, LIMIT=100
  SELECT Account_Number
  FROM Accounts
  WHERE Balance_Current < 500 ;
.EXPORT RESET
.QUIT
```

To execute this script in a UNIX environment:

```
$ bteq < export2.btq | tee export2.out
*** Success, Stmt# 1 ActivityCount = 330
*** Query completed. 330 rows found. 1 column returned.
*** Total elapsed time was 1 second.

*** Warning: RetLimit exceeded.
          Ignoring the rest of the output.

*** Output returned to console
$
```

datafile2 – contains 100 account numbers
export2.out – contains the output informational text sent to the console

BTEQ EXPORT – Example 3

The facing page displays a simple BTEQ .EXPORT script which exports a CSV (Comma Separated Value) file.

You have to specifically export the delimiter (or the comma in a CSV file) for each field that you are exporting. Integer data is cast as variable character data.



BTEQ EXPORT – Example 3

This script exports a data file with fields that are separated by a comma – referred to as a CSV data file (CSV – Comma Separated Value).

export3.btq

```
.LOGON tdp1/user1,passwd1
.OS rm custdata_csv
.EXPORT REPORT FILE = custdata_csv
  SELECT      CAST(Customer_Number    AS VARCHAR(11))      ||','||
               CAST>Last_Name          AS VARCHAR(30))      ||','||
               CAST(First_Name         AS VARCHAR(20))      ||','||
               CAST(Social_Security   AS VARCHAR(9))       (TITLE ")
  FROM        Customer
  SAMPLE      100;
.EXPORT RESET
.QUIT
```

Effectively
exported as 1
concatenated
field.

Examples of exported records are:

```
8062,Graham,Dennis,213629066
3168,Michelson,Mervin,296604596
2327,Green,Sharon,271600391
```

Note: Each record has an EOR terminator (hex '0A') that is automatically added.

Indicator Variables

Teradata invented the word INDICDATA for utilities that submit pure SQL as opposed to SQL-like utility statements. INDICDATA is used for BTEQ to instruct the software that these indicator bytes are present in the data file.

Note: The keyword INDICATORS is used for all the other utilities.



Indicator Variables

Indicator variables allow utilities to process records that contain NULL indicators.

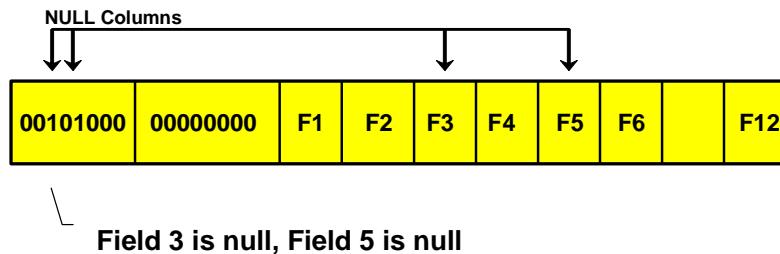
.EXPORT INDICDATA
.SET INDICDATA [ON]

} BTEQ

INDICATORS

} FastLoad
MultiLoad
FastExport
TPump

INDICATORS causes leading n bytes of the record as NULL indicators instead of data.



Determining the Logical Record Length with Fixed Length Columns

Some host systems, such as IBM mainframes, require the correct LRECL (Logical Record Length) parameter in JCL, and will abend if the value is not correctly stated. Other types of hosts are less sensitive to this requirement.

If the input data contains only fixed-length columns, the Logical Record Length is relatively easy to calculate. Each record being treated has the same length. Fixed-length columns must accommodate the largest possible value, however, and frequently involve poor utilization of disk space.

The example on the facing page is in EBCDIC, but even so the wastage of space involved with storage of non-functional blanks (HEX 40) is apparent.

Note: For convenience, HEX representations are provided in EBCDIC only.



Determining the Logical Record Length with Fixed Length Columns

	<u>Length</u>
CREATE TABLE Customer, FALBACK	
(Customer_Number INTEGER 4)	
,Last_Name CHAR(8) 8)	
,First_Name CHAR(8) 8)	
,Social_Security INTEGER 4)	
,Birth_Date DATE 4)	
,OD_Limit DECIMAL(7,2)) 4)	
UNIQUE PRIMARY INDEX (Customer_Number);	
	<u>Total</u> 32

J o n e s															J a c k														
0	0	0	0	D	9	9	8	A	4	4	4	D	8	8	9	4	4	4	4	0	0	0	0	0	0	7	3	...	
0	0	0	6	1	6	5	5	2	0	0	0	1	1	3	2	0	0	0	0	0	0	0	0	0	0	9	A	B	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28		

Customer #
 Last Name
 First Name
 Social Security
 Birth Date

Determining the Logical Record Length with Variable Length Columns

By defining variable character columns as VARCHAR, you can frequently save a significant amount of storage space, but this does have a cost. Each variable-length column is required to provide a 2-byte leading length field, but you no longer have the cost of trailing blanks.

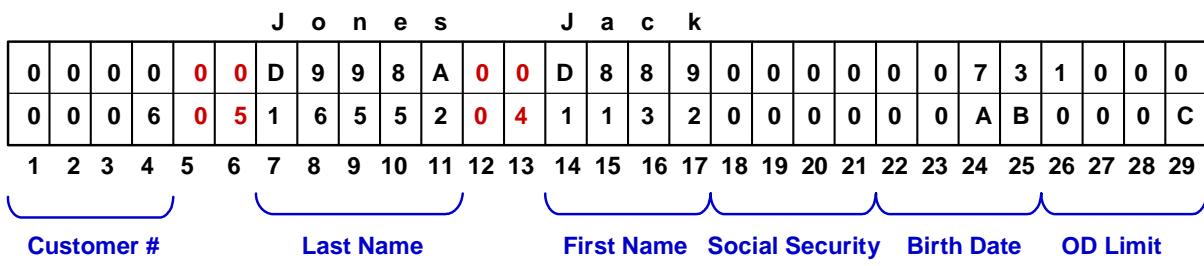
In calculating the correct LRECL parameter, you must allow not only for the 2-byte length field, but also for the *largest* column length accommodated. While the Logical Record Length may grow, the records themselves are typically shorter and of varying length.



Determining the Logical Record Length with Variable Length Columns

```
CREATE TABLE Customer, FALBACK
(Customer_Number      INTEGER          4
,Last_Name            VARCHAR(8)       10
,First_Name           VARCHAR(8)       10
,Social_Security      INTEGER          4
,Birth_Date            DATE             4
,OD_Limit              DECIMAL(7,2)     4
UNIQUE PRIMARY INDEX (Customer_Number);
```

Last_Name and **First_Name** redefined each as **VARCHAR(8)** reduces storage by 7 spaces, but adds two 2-byte length fields.



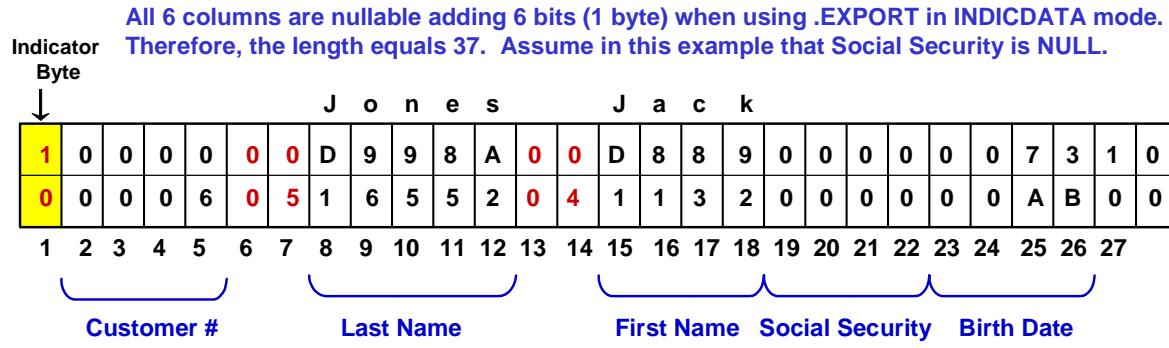
Determining the Logical Record Length with .EXPORT INDICDATA

If Indicator Variables are also used, then 1 byte will be allocated for every 8 columns or fields. For example, if there are 12 columns, then 2 bytes are needed for the NULL bits.



Determining the Logical Record Length with .EXPORT INDICDATA

			<u>Length</u>
CREATE TABLE Customer, FALBACK			
(Customer_Number	INTEGER	4	
,Last_Name	VARCHAR(8)	10	
,First_Name	VARCHAR(8)	10	
,Social_Security	INTEGER	4	
,Birth_Date	DATE	4	
,OD_Limit	DECIMAL(7,2))	4	
UNIQUE PRIMARY INDEX (Customer_Number);			Assume NULL for Social Security
			Total 37



.IMPORT (for Network-Attached Systems)

BTEQ is also useful when you want to IMPORT data from a network-attached server to Teradata as a series of INSERTs, UPDATEs, DELETEs, and “macro” transactions.

Since the Teradata database performs all necessary conversions from displayable characters to binary format, BTEQ .IMPORT has no need to support the concept of FIELD mode.

BTEQ supports DATA, INDICDATA, REPORT, and VARTEXT formats for import. The VARTEXT record format is a Teradata V2R4 feature.

As .EXPORT permits the application programmer to limit the number of records written to the host file, .IMPORT allows you to skip a specified number of records at the beginning of the file. This allows you to derive a more typical sampling of transactions from the middle of the file.

VARTEXT Notes

The following rules apply to VARTEXT records:

1. The only acceptable data types are VARCHAR, VARBYTE, and LONG VARCHAR.
2. A delimiter at the end of the input record is optional.
3. The number of data items in the input record must be equal to the number of fields defined in the USING clause.
4. Two consecutive delimiter characters specify that the corresponding field should be “nulled”. If the record starts with a delimiter, the first field will be “nulled”.
5. You can use SET REPEATSTOP ON to stop inserting if an error is encountered during processing of a VARTEXT record. By default, it rejects the record and continues processing.

AXSMOD Example

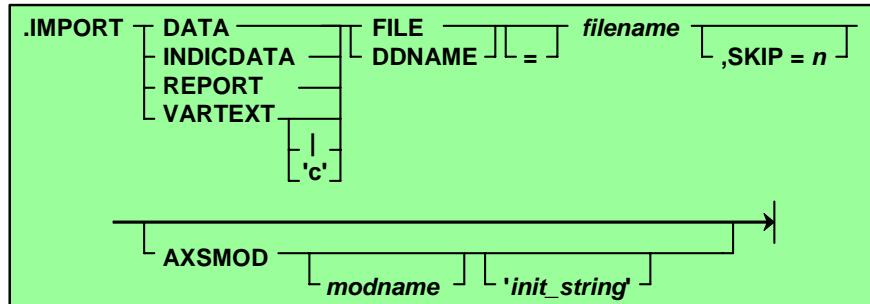
The AXSMOD allows BTEQ to process a data file greater than 2 GB from a tape subsystem. The following two examples show an EXPORT/IMPORT pair with AXSMOD on a Windows 2000 platform using TNTBAR (Backup and Restore).

```
.EXPORT DATA FILE = 'temp' AXSMOD tntbar.dll 'device=tape0 tapeidbteq1 volset=vol1'  
.IMPORT DATA FILE = 'temp' AXSMOD tntbar.dll 'device=tape0 tapeidbteq1 volset=vol1'
```



.IMPORT (for Network-Attached Systems)

IMPORT loads data from a server to the Teradata database with a **USING** clause.



DATA imports data from the server to Teradata with a **USING** clause.

INDICDATA import records contain NULL bits.

REPORT imports Teradata “report” data. Data expected in BTEQ EXPORT REPORT format.

VARTEXT record format as variable length character fields. Default delimiter is | or specify with field delimiter within single quotes.

AXSMOD access module used to import from tape, named pipes, etc.

.IMPORT (for Channel-Attached Systems)

BTEQ is also useful when you want to IMPORT data from a channel-attached host to Teradata as a series of INSERTs, UPDATEs, DELETEs, and “macro” transactions.

Since the Teradata database performs all necessary conversions from displayable characters to binary format, BTEQ .IMPORT has no need to support the concept of FIELD mode. It only supports DATA (flat-file input) and INDICDATA modes for Channel-Attached systems.

As .EXPORT permits the application programmer to limit the number of records written to the host file, .IMPORT allows you to skip a specified number of records at the beginning of the file. This allows you to derive a more typical sampling of transactions from the middle of the file.



.IMPORT (for Channel-Attached Systems)

IMPORT loads data from the host to the Teradata database with a **USING** clause.

INDICDATA preserves nulls.

```
.IMPORT [ DATA [ INDICDATA ] | FILE [ = ] ddname [,SKIP = n ] ]
```

- | | |
|---------------------|--|
| { .IMPORT DATA | Reads a host file in record mode. |
| { .IMPORT INDICDATA | Reads data in host format using indicator variables in record mode to identify nulls. |
| { DDNAME | Name of MVS JCL DD statement or CMS FILEDEF. |
| { FILE | Name of input data set in all other environments. |
| SKIP = n | Number of initial records from the data stream that should be skipped before the first row is transmitted. |

.PACK

The PACK command provides the capability of sending multiple IMPORT data file records along with an SQL request. BTEQ uses this factor to indicate an upper limit when determining how many records to pack into the USING data buffer sent with the SQL request.

A pack factor is established by using the PACK command or by using a PACK clause for the REPEAT command.

.REPEAT

Submits the next Teradata SQL request a specified number of times.

The REPEAT command is typically used with Teradata SQL requests that contain a USING clause. Each time the request is submitted, it uses the next data row from the input data stream. The REPEAT command is cancelled if the input file cannot be accessed.

Options include:

- n* How many times you want to submit the next request.
- * The next request is to be submitted continuously until the import file runs out of data.

RECS *r* Where *r* is an integer in the range of 1 .. 2251636603879500

PACK *p* The REPEAT command's PACK clause overrides the PACK command setting for the duration of the repeat. Once the repeat is over, the pack factor returns to the value associated with the PACK command.

Example1: .REPEAT * PACK 100

This equates to repeat the request as many times as possible before reaching EOF (or max n) and pack up to 100 records with each request. The number of records actually sent is determined at REPEAT completion. The pack factor actually used may vary for each request sent.

Example 2: .REPEAT RECS 200 PACK 100

If you need to ensure an exact number of records are transferred, use the RECS clause version for the repeat factor to compensate for any "reduced" requests. For example,

This equates to repeat the request as many times as necessary to read up to 200 records and pack a maximum of 100 records with each request.



.PACK and .REPEAT

.PACK – Specifies how many records to pack into the USING data buffer sent with the SQL request.

```
.PACK — n —
```

The PACK command provides the capability of sending multiple IMPORT data file records along with an SQL request. BTEQ uses this factor to indicate an upper limit.

.REPEAT – Submits the next Teradata SQL request a specified number of times.

```
.REPEAT [n] [* | RECS r] PACK p
```

The REPEAT command's PACK clause overrides the PACK command setting for the duration of the repeat.

BTEQ IMPORT – Example 1

The facing page displays a sample BTEQ .IMPORT script with several interesting features:

- Elimination of individual accounting output (.QUIET ON)
- .REPEAT

.QUIET ON is used to avoid individual statement accounting in the default output file. If multiple sessions are used, all individual row accounting is eliminated, and final statistics are provided. This method avoids a great deal of performance-limiting I/O on the host.

The .REPEAT statement causes BTEQ to continue reading input values until reaching the limit specified. If this command is omitted, BTEQ will perform the required action only once.

When using .REPEAT:

- The asterisk (*) causes the next request to be submitted continuously until the import file runs out of data.
- You can specify a number to indicate how many times you want to submit the next request.



BTEQ IMPORT – Example 1

(loading data from a UNIX Server)

import1.btq

```
.LOGON tdp1/user1,passwd1
.IMPORT DATA FILE = datafile1a;
.QUIET ON
.REPEAT *
  USING      in_CustNo      (INTEGER)
            ,in_SocSec     (INTEGER)
            ,Filler        (CHAR(30))
            ,in_Lname      (CHAR(20))
            ,in_Fname      (CHAR(10))
  INSERT INTO Customer
    (Customer_Number
     ,Last_Name
     ,First_Name
     ,Social_Security )
  VALUES
    ( :in_CustNo
     ,:in_Lname
     ,:in_Fname
     ,:in_SocSec)
;
.QUIT
```

.QUIET ON

Limits output to reporting only errors and request processing statistics.

.REPEAT *

Causes BTEQ to read records until EOF.

USING

Defines the input data from the host.

To execute:

bteq < import1.btq | tee import1.out

BTEQ IMPORT – Example 2

On a UNIX platform, **bteq** can be called in either its pseudo-interactive mode or by reference to an input script. To specify the script files, use the “<” for input redirection and the “>” for output redirection.



BTEQ IMPORT – Example 2

(using imported data as update data)

```
bteq
Enter your BTEQ Command:
.RUN FILE = c:\td_scripts\import2.btq
or
bteq < c:\td_scripts\import2.btq
```

import2.btq

```
. LOGON tpd1/user1,passwd1
. IMPORT DATA FILE = c:\td_datafiles\datafile2
. QUIET ON
. REPEAT * PACK 10
  USING      in_CustNo          (INTEGER)
             , in_SocSec        (INTEGER)
  UPDATE    Customer
  SET       Social_Security   = :in_SocSec
  WHERE    Customer_Number   = :in_CustNo
;UPDATE   Customer_History
  SET       Social_Security   = :in_SocSec
  WHERE    Customer_Number   = :in_CustNo ;
.QUIT;
```

This example shows execution of a BTEQ script on a Windows server and the optional use of the .RUN command.

.RUN – processes the Teradata SQL requests and BTEQ commands from the specified run file.

.REPEAT * PACK 10

Causes BTEQ to read records until EOF. Up to 10 import records are sent along with the SQL request.

Note: Multi-statement request

Allows BTEQ to use imported data with multiple tables.

BTEQ IMPORT – Example 3

The facing page displays a simple BTEQ .IMPORT script which imports a CSV (Comma Separated Value) file.

You have to specify the delimiter (the comma in a CSV file) since the default is the |.



BTEQ IMPORT – Example 3

(importing data from a CSV file)

This script effectively imports data from a file with fields that are separated by a comma; referred to as a CSV data file (CSV – Comma Separated Value).

export3.btq

```
.LOGON tdp1/user1,passwd1
.IMPORT VARTEXT ',' FILE = custdata_csv
.QUIET ON
.REPEAT *
  USING ( in_custno      VARCHAR(11),
          in_Iname       VARCHAR(30),
          in_fname       VARCHAR(20),
          in_ssn        VARCHAR(9) )
  INSERT INTO Customer
    VALUES (:in_custno, :in_Iname, :in_fname,:in_ssn);
.QUIET OFF
SELECT COUNT(*) FROM Customer;
.QUIT
```

To execute:

bteq < export3.btq | tee export3.out

Multiple Sessions

A session might be defined as a “logon to the Teradata database that permits the user to single-thread one or more sequential transactions that continue until a LOGOFF statement is sent.”

Teradata interprets multiple sessions as a number of users logging on (with the same user ID and password), with each one sending sequential transactions until it submits a LOGOFF statement.

Multiple sessions permit the Teradata database to work on multiple tasks in parallel. This capability requires special software in the (sequential processing) host to enable it to send *multiple requests at the same time*. This special software is the Call Level Interface.

For multiple sessions to be effective in parallel, the system uses *row hash locks* rather than *full table locks*. Since the only type of access that uses row hash locking is the Primary or Unique Secondary Index request, multiple sessions are useful only in connection with UPI, NUPI or USI transactions. All other access requires a full table lock and results in sequential access to tables.



Multiple Sessions

- **Session:**
 - Logical connection between host and Teradata database.
 - Work stream composed of a series of requests between the host and the database.
- **Multiple sessions:**
 - Allow tasks to be worked on in parallel.
 - Require row hash locking for parallel processing: UPI, NUPI, USI transactions.
 - Too few degrade performance.
 - Too many will not improve performance.
- **Initializing a single session typically takes 1 to 2 seconds.**

.SET SESSIONS

The .SET SESSIONS command must appear prior to the .LOGON statement since it directs BTEQ to initialize the appropriate number of LOGONs. The SESSIONS parameter tells BTEQ how many times to log on to the Teradata database for parallel access.

The sessions are then logged on by TDP or the UNIX GATEWAY and logged on to the dedicated PEs to distribute the workload as evenly as possible.

If only a single session (no .SET SESSION statement) is used, BTEQ will still provide the elapsed time associated with each DML.

In Teradata V2R4.1 (and before), the limit on the number of sessions that you can request with BTEQ depends on the operating environment.

DOS/Windows – 16

UNIX – 20

VM/MVS – 200

In Teradata V2R5, the limit on the number of sessions that you can request with BTEQ is 200 for all operating environments.

Typically, BTEQ Imports will take advantage of multiple sessions and BTEQ Exports will not.



.SET SESSIONS



```
.SET SESSIONS 8
.LOGON tdp1/user1,passwd1
.IMPORT DATA FILE = datafile4
.QUIET ON
.REPEAT *
USING    in_CustNo      (INTEGER)
          , in_SocSec     (INTEGER)
          , Filler        (CHAR(30))
          , in_LName      (CHAR(20))
          , in_FName      (CHAR(10))

INSERT INTO Customer
( Customer_Number
, Last_Name
, First_Name
, Social_Security )
VALUES ( :in_CustNo
        ,:in_LName
        ,:in_FName
        ,:in_SocSec )
;
.QUIT
```

Parallel Processing Using Multiple Sessions to Access Individual Rows

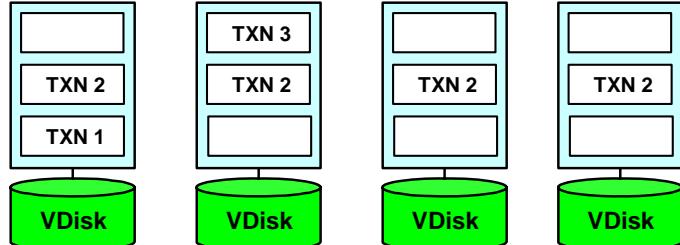
It is important to remember that multiple sessions only benefit transactions that do *not* use a full table lock.

If multiple sessions are specified where a full table lock is required, not only will they execute sequentially but performance will be inhibited by the system resources needed for management.

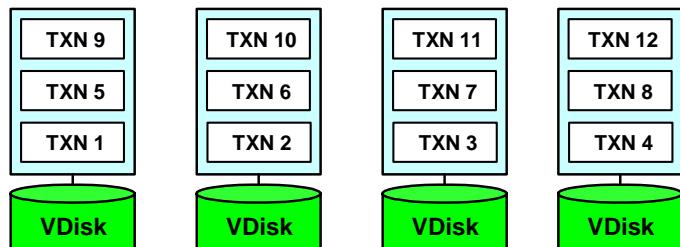


Parallel Processing Using Multiple Sessions to Access Individual Rows

Although a single row can reside on 1 AMP, it might require a full table scan to locate it. All AMPs are required to participate.



If the location of the row is known, only the applicable AMP needs to be involved. Other AMPs can work on other tasks — multiple sessions are useful.



Multiple transactions execute in parallel, provided that:

- Each transaction uses fewer than all AMPs.
- Enough are sent to keep ALL AMPs busy.
- Each parallel transaction has a unique internal ID.

When Do Multiple Sessions Make Sense?

Assuming there are no fallback tables:

- Primary Index transactions use 1 AMP and a Row Hash lock.
- Unique Secondary Index transactions use 2 AMPS and Row Hash locks.
- Non-unique Secondary Index transactions and full table scans use all AMPS and table-level locks.

You should now be in a position to respond to users who complain that they were using multiple sessions to perform BTEQ updates, that the job ran faster with one session, and that BTEQ was broken.

Try to complete the table on the facing page to determine which type of requests would benefit from multiple sessions.



When Do Multiple Sessions Make Sense?

Multiple sessions improve performance ONLY for SQL requests that impact fewer than ALL AMPs.

TRANS_HISTORY

Trans_Number	Trans_Date	Account_Number	Trans_ID	Amount
PK		FK,NN		
USI		NUPI		
NUSI				

Which of the following batch requests would benefit from multiple sessions?

1. **INSERT INTO Trans_History
VALUES (:T_Nbr, DATE, :Acct_Nbr, :T_ID, :Amt);**
2. **SELECT * FROM Trans_History
WHERE Trans_Number=:Trans_Number;**
3. **DELETE FROM Trans_History
WHERE Trans_Date < DATE - 120;**
4. **DELETE FROM Trans_History
WHERE Account_Number= :Account_Number;**

Trans Type	Table or Row Lock	Multiple Sessions Useful or Not?

Application Utility Checklist

The checklist on the facing page summarizes BTEQ functions. It will be completed for each utility as it is discussed.

While BTEQ efficiently restarts from a failure of the Teradata database system, it must rollback in the event of a host failure and restart from the first record.

Clearly this can be a distinct disadvantage. While BTEQ can be very useful and performs reasonably well, your applications probably also require the ability to restart from a host failure without rolling back to the first record.

For this reason, many of the other application utilities have a restart capability built in.



Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL				
DML Functions	ALL				
Multiple DML	Yes				
Multiple Tables	Yes				
Multiple Sessions	Yes				
Protocol Used	SQL				
Conditional Expressions	Yes				
Arithmetic Calculations	Yes				
Data Conversion	Yes				
Error Tables	No				
Error Limits	No				
User-written Routines	No				
Uses Support Environment	No				

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Answer True or False.

1. True or False. With BTEQ you can import data from the host to Teradata AND export from Teradata to the host.
2. True or False. .EXPORT DATA sends results to a host file in field mode.
3. True or False. INDICDATA is used to preserve nulls.
4. True or False. With BTEQ, you can use conditional logic to bypass statements based on a test of an error code.
5. True or False. It is useful to employ multiple sessions when ALL AMPS will be used for the transaction.
6. True or False. With .EXPORT, you can have output converted to a format that can be used with PC programs.

Lab Exercise 33-1

After creating the data file named **data33_1**, you may want to check its size to ensure that you have created it correctly. An easy way to do this is in UNIX is to use the UNIX **ls -l** command.

The size of this file (data33_1) should be 312,000 bytes.

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab33_14.btq** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function

To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 33-1

Purpose

In this lab, you will use BTEQ to perform imports with different numbers of sessions. You will move selected rows from the AP.Accounts table to your personal Accounts table and from a data file to your table. You will repeat tasks using different numbers of sessions.

What you need

Populated AP.Accounts table and your empty Accounts table.

Tasks

1. INSERT/SELECT all rows from the populated AP.Accounts table to your own “userid.Accounts” table. Note the timing and verify that you have the correct number of rows.
Time: Number of rows:
2. Export 4000 rows to a data file (data33_1).
3. Delete all rows from your “userid.Accounts” table.
4. Import the rows from your data set (data33_1) to your empty “userid.Accounts” table. Note the time and verify the number of rows.
Time: Number of rows:
5. Delete all the rows from your “userid.Accounts” table again.

Teradata Training

Lab Notes



Lab Exercises (cont.)

Lab Exercise 33-1 (cont.)

Tasks

6. Specify 8 sessions and import the rows from your data set to your empty “userid.Accounts” table. Note the time and verify the number of rows.

Time: Number of rows:

7. Delete all the rows from your “userid.Accounts” table again.

8. Specify 8 sessions and use a PACK of 10 and import the rows from your data set to your empty “userid.Accounts” table. Note the timing and verify the number of rows.

Time: Number of rows:

9. What are your conclusions based on the tasks you have just performed?

10. If you specified 210 sessions and attempted to import the rows from your data set to your empty “userid.Accounts” table., how many sessions do you think you will be given? _____

Lab Exercise 33-2

The size of data33_2 should be 3,500 bytes.

Note: If your data file size is 30,500 bytes, you exported all of the columns from the Customer table instead of just the Customer Number.



Lab Exercises (cont.)

Lab Exercise 33-2

Purpose

In this exercise, you will use BTEQ to select 500 rows from the AP.Customer table, representing a specific set of 500 customers. First, you will use .EXPORT DATA to build a data set that contains 500 customer numbers and use this as input to access the Customer table. and use .EXPORT REPORT to generate a report file.

What you need

Populated AP.Customer table

Tasks

1. From the AP.Customer table, export to a data file (data33_2) the 500 customer numbers for the customers that have the highest Social Security numbers. (Hint: You will need to use descending order for Social Security numbers.)
2. Using the 500 customer numbers (in data33_2) to select the 500 appropriate rows from AP.Customer, export a report file named "report33_2". In your report you will need the fields: Customer_Number, Last_Name, First_Name, Social_Security.

Hint: You will .IMPORT DATA from data33_2 and use .EXPORT REPORT to report33_2.

3. View your report. The completed report should look like this:

<u>Customer Number</u>	<u>Last Name</u>	<u>First Name</u>	<u>Social Security</u>
2001	Smith	John	123456789

4. What are highest and lowest Social Security numbers in your report?

Highest: _____ Lowest: _____

Teradata Training

Notes

Module 34



FastLoad

After completing this module, you will be able to:

- **Describe the two phases of FastLoad.**
- **Prepare a FastLoad script.**
- **Partition a data load over successive runs.**
- **Restart an interrupted FastLoad.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

FastLoad.....	34-4
FastLoad Characteristics	34-6
FastLoad Phase 1	34-8
FastLoad Phase 2	34-10
A Sample FastLoad Script	34-12
Converting the Data	34-14
Data Conversion Chart.....	34-16
NULLIF	34-18
FastLoading Zoned Decimals and Time Stamps	34-20
FastLoad BEGIN LOADING Statement	34-22
BEGIN LOADING Statement	34-22
FastLoad Error Tables.....	34-24
Error Recovery.....	34-26
CHECKPOINT Option	34-28
END LOADING Statement	34-30
RECORD Statement	34-32
INSERT Statement.....	34-34
Staged Loading of Multiple Data Files	34-36
FastLoad Fails to Complete	34-38
Restarting FastLoad (Output).....	34-40
Restarting FastLoad – Summary	34-42
Additional FastLoad Commands	34-44
FastLoad with Additional Options.....	34-46
Invoking FastLoad	34-48
INMOD	34-50
Application Utility Checklist	34-52
Summary	34-54
Review Questions	34-56
Lab Exercise 34-1	34-58
Lab Exercise 34-2	34-60

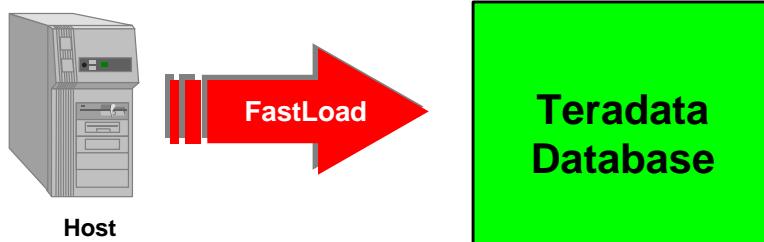
FastLoad

The facing page describes the capabilities of FastLoad.



FastLoad

- **Fast batch mode utility for loading empty tables in the Teradata database**
 - Tables cannot have any secondary, join, or hash indexes
- **Full Restart capability**
- **Error Limits and Error Tables, accessible using SQL**
- **Restartable INMOD routine capability**
- **Ability to load data in several stages**



FastLoad Characteristics

The FastLoad utility can be executed as a client utility in UNIX, Windows, Linux, and in the mainframe environments.



FastLoad Characteristics

Purpose

- Load large amounts of data into an empty table at high speed.
- Execute from servers, channel, or network-attached hosts.

Concepts

- Loads into an empty table with no secondary, join, or hash indexes.
- Has two phases – creates an error table for each phase.
- Status of run is displayed.
- Checkpoints can be taken for restarts.

Restrictions

- Only load 1 empty table with 1 FastLoad job.
- The system parameters **MaxLoadTasks** and **MaxLoadAWT** determine the maximum number of FastLoad jobs that can run at one time.
- FastLoad cannot load data into tables defined with Referential integrity, Secondary Indexes, Join Indexes, Hash Indexes, or Triggers.
 - Tables with Soft Referential Integrity can be loaded with FastLoad.
- Duplicate rows cannot be loaded into a multiset table with FastLoad.
- If an AMP goes down, FastLoad cannot be restarted until the AMP is back online.

FastLoad Phase 1

FastLoad has two recognizable phases. These can be called Phase 1 (Acquisition Phase) and Phase 2 (Insert or Application Phase).

Note: The blocks can be up to 63.5 KB (default).

The illustration on the facing page shows the process used in Phase 1.

Step 1 – FastLoad is executed on a host and sends blocks of records to PE sessions.

Step 2 – Parsing Engine Sessions receive a block of records from the FastLoad utility and simply passes the block to an AMP via the BYNET.

Step 3 – The AMP receives a block of records in memory.

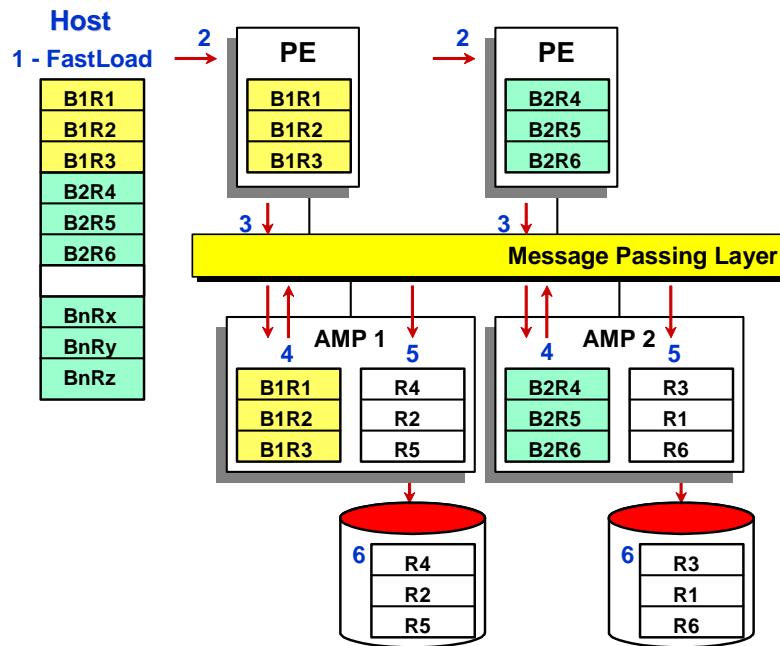
Step 4 – The AMP hashes each record in the block and redistributes each row to the Message Passing Layer (PDE and BYNET).

Step 5 – The Message Passing Layer delivers rows to the appropriate AMP based on row hash value. Each AMP collects the rows in memory.

Step 6 – When enough rows are collected to fill a block, the AMP writes the block to disk.

At this point, the AMP has the rows it should have, but they are not in row hash sequence.

FastLoad Phase 1



Phase 1

- FastLoad uses one SQL session to define AMP steps.
- The PE sends a block to each AMP which stores blocks of unsorted data records.
- AMPs hash each record and redistribute them to the AMP responsible for the hash value.
- At the end of Phase 1, each AMP has the rows it should have, but the rows are not in row hash sequence.

FastLoad Phase 2

The second phase of FastLoad has each AMP (in parallel) reading the data blocks from disk, sorting the data rows based on row hash, and writing the blocks back out to PERM space.

The illustration on the facing page shows the process used in Phase 2.

Step 1 – FastLoad receives the END LOADING; statement.

Step 2 – FastLoad sends a request to the Parsing Engine to indicate the start of Phase 2.

Step 3 – The PE broadcasts the start of Phase 2 to all AMPs.

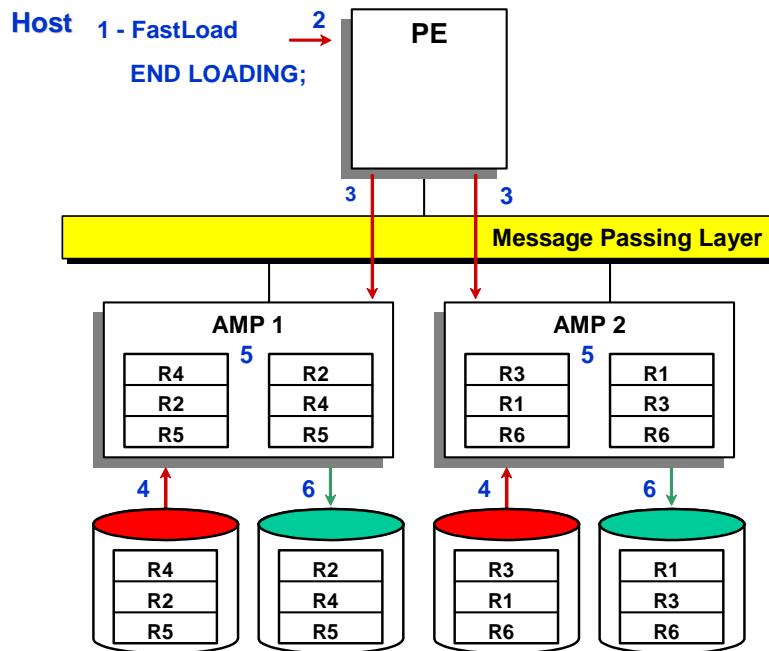
Step 4 – Each AMP reads its blocks in from disk.

Step 5 – Each AMP sorts its data rows based on row hash sequence.

Step 6 – Each AMP writes the sorted blocks back to disk.

If the table is Fallback protected, then the Fallback copy of data is created at this time. The table is made available for user access.

FastLoad Phase 2



Phase 2

- When the FastLoad job receives END LOADING; statement, FastLoad starts Phase 2.
- Each AMP sorts the target table, puts the rows into blocks, and writes the blocks to disk.
- Fallback rows are then generated if required.
- Table data is available when Phase 2 completes.

A Sample FastLoad Script

The job on the facing page first cleans up and prepares the environment for use. It could be performed using BTEQ.

The first step is to make sure that the table is empty and to remove old error tables before starting the FastLoad job.

Often it is better to perform set-up steps outside the FastLoad script so that the FastLoad operation can be isolated to perform loading tasks. If a restart is necessary, you will not need to change your FastLoad script to remove the unneeded statements.

The load job on the facing page is being run on a UNIX system as noted by the “FILE=” parameter. If this job was going to be run on a MVS or an OS/390 system, then use the “DDName=” parameter instead of the “File” parameter.



A Sample FastLoad Script

SETUP
Create the table, if it doesn't already exist.

```
LOGON tdpid/username,password;
DROP TABLE Acct;
DROP TABLE AcctErr1;
DROP TABLE AcctErr2;

CREATE TABLE Acct, FALBACK (
    AcctNum      INTEGER
    ,Number      INTEGER
    ,Street       CHAR(25)
    ,City         CHAR(25)
    ,State        CHAR(2)
    ,Zip_Code     INTEGER)
UNIQUE PRIMARY INDEX (AcctNum);
LOGOFF;
```

```
LOGON tdpid/username,password;
BEGIN LOADING Acct
    ERRORFILES AcctErr1, AcctErr2
    CHECKPOINT 100000;
```

Start the utility.
Error files must be defined.

```
DEFINE in_AcctNum (INTEGER)
    ,in_Zip      (INTEGER)
    ,in_Nbr      (INTEGER)
    ,in_Street   (CHAR(25))
    ,in_State   (CHAR(2))
    ,in_City    (CHAR(25))
FILE=data_infile1;
```

Checkpoint is optional.

```
INSERT INTO Acct VALUES (
    :in_AcctNum
    ,:in_Nbr
    ,:in_Street
    ,:in_City
    ,:in_State
    ,:in_Zip);
```

DEFINE the input;
must agree with host data format.

```
END LOADING;
LOGOFF;
```

INSERT must agree with table definition.
Phase 1 begins.
Unsorted blocks are written to disk.

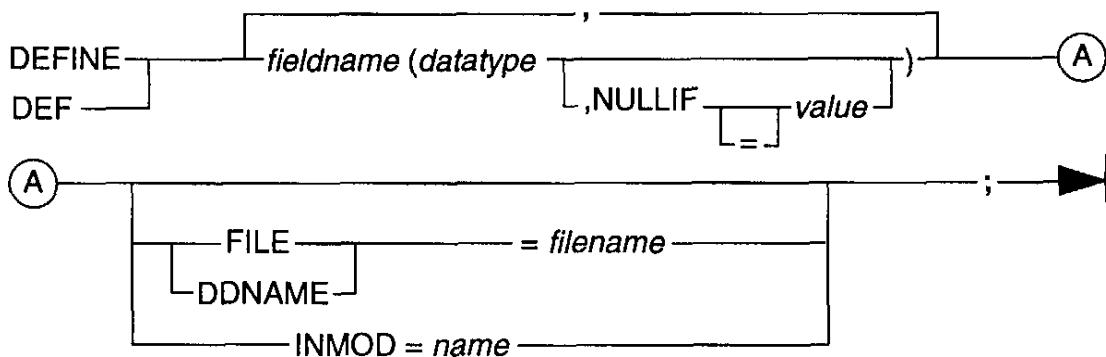
Phase 2 begins with END LOADING. Sorting and writing blocks to disk.

Converting the Data

This example shows two examples of conversions you can perform in the FastLoad environment.

Each input data field (DEFINE) must undergo a conversion to fit in the database field (Create Table).

All are valid conversions and are limited to **one per column**.



Valid data types for table that can be loaded with FastLoad are:

- BIGINT
- BYTE
- BYTEINT
- CHARACTERS(n)
- DATE
- DECIMAL(x) OR DECIMAL(x,y)
- FLOAT
- GRAPHIC(n)
- INTEGER
- LONG VARCHAR
- LONG VARGRAPHIC
- SMALLINT
- VARBYTE(n)
- VARCHAR(n)
- VARGRAPHIC(n)



Converting the Data

```

LOGON educ2/user14,ziplock;
DROP TABLE Accounts;
DROP TABLE Accts_Err1;
DROP TABLE Accts_Err2;
CREATE TABLE Accounts, FALBACK (
    Account_Number INTEGER
    ,Account_Status CHAR(15)
    ,Trans_Date DATE
    ,Balance_Forward DECIMAL(5,2)
    ,Balance_Current DECIMAL(7,2) )
    UNIQUE PRIMARY INDEX
        (Account_Number);
LOGOFF;

```

```

LOGON educ2/user14,ziplock;
BEGIN LOADING Accounts
    ERRORFILES Accts_Err1, Accts_Err2 ;
    DEFINE in_Acctno      (CHAR(9))
          ,in_Trnsdate   (CHAR(10))
          ,in_Balcurr     (CHAR(7))
          ,in_Balfwd      (INTEGER)
          ,in_Status       (CHAR(10))
    FILE = infile_name;
    INSERT INTO Accounts
        (Account_Number
        ,Account_Status
        ,Trans_Date
        ,Balance_Forward
        ,Balance_Current)
    VALUES (
        :in_Acctno
        ,:in_Status
        ,:in_Trnsdate (Format 'YYYY-MM-DD')
        ,:in_Balfwd
        ,:in_Balcurr);
END LOADING;
LOGOFF;

```

Notes:

- FastLoad permits conversion from one data type to another, *once for each column*.
- Including optional column names with the **INSERT** statement provides script documentation which may aid in the future when debugging or modifying the job script.

Data Conversion Chart

On the facing page is a comprehensive chart of possible conversions, showing the old data type and the new data type, and sample data for each.

An INVALID result comes from an unsupported conversion.

An overflow output is the result of too much data for the receiving field.

General Notes:

- The “target table” can have range constraints” defined at the column level when the table is created and these will be checked as part of Phase 1 with FastLoad.
- If the input field has leading or trailing spaces (blanks) and you are converting to a numeric field (e.g., INTEGER), leading/trailing spaces are ignored.

Ex.	converted to
'0001'	0001
' 001'	0001
'001 '	0001
' 01 '	0001
' '	0000
'1 '	0001
'1 0'	conversion error



Data Conversion Examples

FROM:	TO:	ORIGINAL DATA:	STORED AS:
CHAR(13)	VARCHAR(5)	ABCDEFGHIJKLM	ABCDE
CHAR(5)	INTEGER	ABCDE	invalid
CHAR(5)	INTEGER	12345	0000012345
CHAR(13)	INTEGER	12345bbbbbbb	0000012345
CHAR(13)	INTEGER	1234567890123	overflow
CHAR(13)	DATE	92/01/15bbbbbb	920115
CHAR(13)	DATE	920115bbbbbbb	invalid
CHAR(13)	DATE	01/15/92bbbbbb	invalid
CHAR(6)	DEC(5,2)	123.50	123.50
CHAR(6)	DEC(5,2)	12350	overflow
VARCHAR(5)	CHAR(13)	ABCDE	ABCDEbbbbbbbb
BYTEINT	INTEGER	123	0000000123
SMALLINT	INTEGER	12345	0000012345
INTEGER	SMALLINT	0000012345	12345
INTEGER	SMALLINT	1234567890	invalid
INTEGER	BYTEINT	0000000123	123
INTEGER	BYTEINT	0000012345	invalid
INTEGER	DATE	0000920115	920115
INTEGER	CHAR(8)	0000012345	bbb12345
DECIMAL(3,2)	INTEGER	1v23	0000000001
DECIMAL(3,2)	CHAR(5)	1v23	b1.23
DECIMAL(3,2)	CHAR(3)	1v23	1.2
DATE	INTEGER	0000920115	0000920115
DATE	SMALLINT	0000920115	invalid
DATE	CHAR(8)	0000920115	92/01/15
DATE	CHAR(6)	0000920115	92/01/

NULLIF

NULLIF is used for special conversions. When another system uses a special combination to represent unknown data values, you can test for them and convert them to NULL in the Teradata database.

If you attempt to place a zero (0) into a DATE field, you will get the following error:

#3520 A constant value in a query is not valid for column *colname*.



NULLIF

```
DEFINE  in_Acctno      (CHAR(9))
       ,in_Status       (CHAR(10))
       ,in_Trnsdate    (CHAR(10), NULLIF = '0000-00-00')
       ,in_Balfwd       (INTEGER)
       ,in_Balcurr     (CHAR(7))
FILE = infile5;

INSERT INTO Accounts VALUES (
       :in_Acctno
       ,:in_Status
       ,:in_Trnsdate   (FORMAT 'YYYY-MM-DD')
       ,:in_Balfwd
       ,:in_Balcurr);
```



- **NULLIF** allows you to specify that if an input field contains a specified value, it should be treated as **NULL**.
- One common example occurs when dates are entered as zeroes; they may cause a fault since they are not in the expected format.

FastLoading Zoned Decimals and Time Stamps

On the facing page are two common conversion situations that you may encounter.

“Packed Decimal” is a mainframe data type that can be converted to decimal in the Teradata database.

Dates and/or Time Stamps are often presented to the loader in display form or character format. To convert them into acceptable dates or time stamps in the database, you must identify the input form with a “format” statement.

The following are acceptable in DATE FORMAT statements:

- yyyy (four digit year)
- yy (year in two digits)
- mmm (three character abbreviation of month)
- mm (two digit month)
- ddd (day of the year)
- dd (day of the month)
- a series of punctuation characters
 - decimal
 - comma
 - dash
 - **b** blank (space)
 - / slash

If the input file has a field with a TimeStamp(0) in it, then

DEFINE input field as CHARACTER (19);
on the INSERT use (FORMAT 'YYYY-MM-DDbHH:MI:SS')

Note: TimeStamp(0) indicates that there are no decimal digits associated with the seconds portion of the time stamp.



FastLoading Zoned Decimals and Time Stamps

- If EBCDIC **unpacked decimal values** are presented for loading into a decimal-type column, an error is returned:

Unpacked

F1	F2	F3	F4	F5	F6	F7	C8
----	----	----	----	----	----	----	----

Decimal (8)

1	2	3	4	5	6	7	H
---	---	---	---	---	---	---	---

Packed decimal

01	23	45	67	8C
----	----	----	----	----

Loads into decimal-defined column with no errors.

2679 Format or data contains a bad character.

- Use the following script structure if a **signed zoned decimal data representation** is required or if **time stamps** are in **character** format.

```

CREATE TABLE Trans
  ( Tr_Num           DECIMAL(8)
  , Tr_DateTime      TIMESTAMP(0)
  ....              );
  DEFINE  in_TNbr      (CHAR(8))
  ,in_TDateTime     (CHAR(19))
  DDNAME=Data1 ;
  INSERT INTO TRANS VALUES
  (:in_TNbr        (FORMAT '9(8)S')
  ,:in_TDateTime   (FORMAT 'YYYY-MM-DDbHH:MI:SS')
  .... );

```

Define unpacked decimal as CHAR data and FORMAT.

FastLoad BEGIN LOADING Statement

The general syntax for the FastLoad statement to begin loading is shown on the facing page.

The BEGIN LOADING statement must reference a table, not a view.

BEGIN LOADING Statement

Use the BEGIN LOADING statement to identify the table that is to be loaded with FastLoad. The table may be in the user's default database (shown), or, by prefixing the name with a *databasename* and a dot (.), you may specify another database.

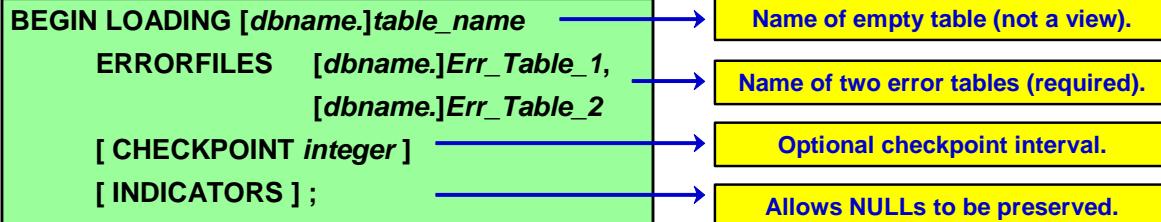
Use the BEGIN LOADING statement to specify (and create) the two error tables for the operation. You can specify error tables in the same database as the data table or in a different database.

You can also define a checkpoint interval for the job. It is specified as a number of input records. The checkpoint will be rounded to the nearest number of records that can fit within a block.

The FastLoad utility can recognize the indicator bytes placed in front of the data record by the application that created the input. These bytes identify the fields in the input record that should be NULL in the created record of the table and can be automatically generated by either BTEQ Export (INDICDATA) or FastExport (INDICATORS).



BEGIN LOADING Statement



A userid needs the following privileges in order to execute the FastLoad utility.

- For *Target_table_name*: SELECT and INSERT (CREATE and DROP or DELETE, if using those functions)
- For *Error_tables*: CREATE TABLE
- Required privileges for the user PUBLIC on the restart log table (SYSADMIN.FASTLOG):
 - SELECT INSERT UPDATE DELETE
- There will be a row in the FASTLOG table for each FastLoad job that has not completed in the system.

FastLoad Error Tables

FastLoad requires two error tables that you specify in the Begin Loading statement. Error tables capture data and duplication errors.

FastLoad discards empty error tables at the successful conclusion of the loader job. If they contain data, error tables are maintained for you to use in analyzing errors.

You must remove the error tables before you re-run the same load job or it will terminate in an error condition.

If you must restart a FastLoad job, the error tables must already exist.



FastLoad Error Tables

ErrorTable1

Contains one row for each row which failed to be loaded due to **constraint violations or translation errors**. The table has three columns:

Column_Name	Datatype	Content
ErrorCode	Integer	The Error Code in DBC.ErrorMsgs.
ErrorFieldName	VarChar(30)	The column that caused the error.
DataParcel	VarByte(64000)	The data record sent by the host.

ErrorTable2

For non-duplicate rows, captures those rows that cause a UPI duplicate violation.

Notes

- Duplicate rows are counted and reported but not captured.
- Error tables are automatically dropped if empty upon completion of the run.
- Performance Note: Rows are written into error tables one row at a time. Errors slow down FastLoad.

Error Recovery

The facing page contains some sample output from a FastLoad job. It describes the error situations encountered during the previous FastLoad job.

Because there are errors in both of the error tables, they will be retained by the system for analysis.

You must remember to analyze these error tables and remove them from the system prior to running this same FastLoad script again.



Error Recovery

Output report from FastLoad

A Total Records Read = 35000
B Total ErrorTable 1 = 1250 (Not loaded due to error)
C Total ErrorTable 2 = 30 (Duplicate UPIs only)
D Total Inserts Applied = 33700
E Total Duplicate Rows = 20

$$(A = B + C + D + E)$$

Investigating the failed rows

```
SELECT DISTINCT ErrorCode, ErrorFieldName FROM Error_Table_1;
```

Investigating the duplicate index violations

```
SELECT * FROM Error_Table_2;
```

CHECKPOINT Option

The CHECKPOINT option defines points in a job where FastLoad pauses to record that Teradata has processed a specified number of input records. When you use checkpoints, you do not have to rerun the entire FastLoad job if it stops before completion. FastLoad will use the checkpoint information in the restart log table to determine the restart location.

If you are going to use the CHECKPOINT option, the Reference Manual recommendation is:

For smaller Teradata Database systems,

If records < 4K, then use CHECKPOINT 100,000

If records \geq 4K, then use CHECKPOINT 50,000

For larger Teradata Database systems, increase the CHECKPOINT value.

Because checkpoints slow down FastLoad processing, it is also recommended to set the CHECKPOINT value to effectively take a checkpoint every 10 to 15 minutes. Frequently, this means setting the CHECKPOINT value to a much larger value.



CHECKPOINT Option

BEGIN LOADING . . .
CHECKPOINT *integer*;

- Used to verify that rows have been transmitted and processed.
- Specifies the number of rows transmitted before pausing to take a checkpoint and verify receipt by AMPs.
- If the CHECKPOINT parameter is not specified, FastLoad takes checkpoints as follows:
 - Beginning of Phase 1
 - Every 100,000 input records
 - End of Phase 1
- FastLoad can be restarted from previous checkpoint.
- **Performance Note:** Checkpoints slow down FastLoad processing – set the CHECKPOINT large enough that checkpoints are taken every 10 to 15 minutes. Usually, this requires a CHECKPOINT value much larger than 100,000.

END LOADING Statement

The End Loading statement signifies to the loader that all data has been acquired. At this time the loader can wrap up Phase One and get started with Phase Two.



END LOADING Statement

END LOADING ;

- Indicates that all data rows have been transmitted.
- Begins Phase 2 processing.
- Omission implies:
 - The load is incomplete and will be restarted later.
 - This causes the table that is being loaded to become “FastLoad paused.”
 - If you attempt to access a table (via SQL) that is in a “FastLoad paused” state, you will get the following error.

Error #2652 Operation Not Allowed tablename is being loaded

RECORD Statement

Use the RECORD statement to override any default positioning assumed by the loader.

You can specify the record of the input file to begin with, and optionally, the record to end with.

The RECORD statement is a separate statement that follows the BEGIN LOADING statement and is specified before the DEFINE statement.



RECORD Statement

RECORD [integer] [THRU integer];

- If you do not use a RECORD command, FastLoad reads from the first record in the data source to the last record (or from the last CHECKPOINT).
- RECORD allows control over which input records are to be brought in for loading.
- RECORD is a separate statement used before the DEFINE statement.

Examples:

RECORD 1 THRU 1000;

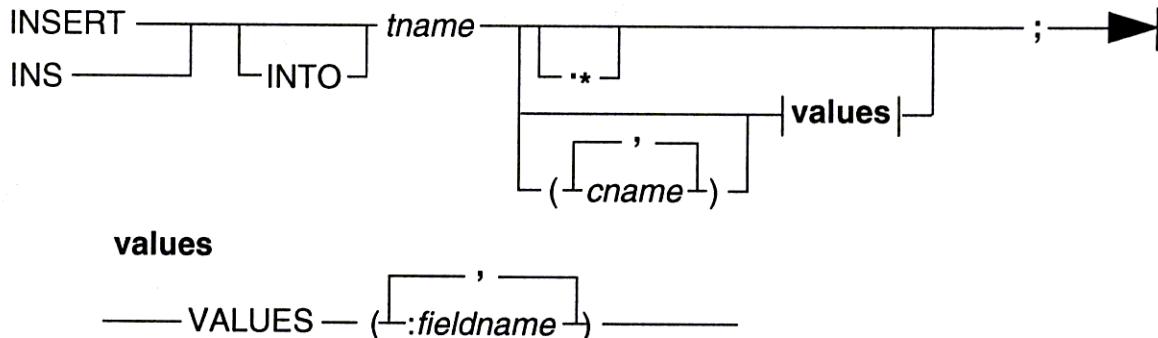
1st through the 1,000th record

RECORD 1;

1st through the last record

INSERT Statement

INSERT is a Teradata SQL statement that inserts data records into the rows of the FastLoad table.



During the insert operation, field values are inserted in the table in the order in which the columns are listed in the CREATE TABLE statement. If field values in the input data are stored in the same order as columns are defined in the CREATE TABLE statement for the FastLoad table, you do not need to specify a list of column names in the INSERT statement.

When you use the wildcard format of the INSERT statement, a list of field names is constructed from the definition of the table. During the insert operation, field names and their data types are taken from the CREATE TABLE statement and used to define the table.

The field name definitions are established in the order in which columns are defined in the CREATE TABLE statement. So, the fields in each data record must be in the same order as the columns in the definition of the table.

When using the second form of the INSERT statement, you still need to use the DEFINE command to specify the name of the input data source or INMOD routine used in the FastLoad job.

If you enter a DEFINE command that defines one or more fields before the INSERT statement, the FastLoad utility appends the field definitions to the definitions constructed from the INSERT statement.

The command example on the following page defines the “**data_file3**” input data source and fields in each record (Account_Number, Number, Street, City, State, and Zip_Code).



INSERT Statement

```

DEFINE      Account_Number  (INTEGER)
              ,Number        (INTEGER)
              ,Street         (CHAR(25))
              ,City          (CHAR(25))
              ,State         (CHAR(2))
              ,Zip_Code      (INTEGER)
FILE = data_file3 ;
INSERT INTO Accounts
              (Account_Number
               ,Number
               ,Street
               ,City
               ,State
               ,Zip_Code)
VALUES
              (:Account_Number
               ,:Number
               ,:Street
               ,:City
               ,:State
               ,:Zip_Code);

```

The “wildcard” format may be used to construct the names in the INSERT statement. The field names are constructed from the DD/D table definition.

```
DEFINE FILE = data_file3 ;
INSERT INTO Accounts.* ;
```

Staged Loading of Multiple Data Files

The example on the facing page illustrates using FastLoad to load two different data sets into a single table.



Staged Loading of Multiple Data Files

load_US.fld

```
LOGON tdpid/username,password;
BEGIN LOADING Customer
    ERRORFILES CustErr1, CustErr2;

DEFINE in_CustNum (INTEGER)
    ,in_Lname (CHAR(15))
    ,in_Fname (CHAR(10))
    ,in_Mailcode (INTEGER)
FILE=US.dat;

INSERT INTO Customer VALUES (
    :in_CustNum
    ,:in_Lname
    ,:in_Fname
    ,:in_Mailcode);

LOGOFF;
```

No END LOADING statement. Table is in “FastLoad Paused” state.

fastload < load_US.fld

load_Int.fld

fastload < load_Int.fld

```
LOGON tdpid/username,password;
BEGIN LOADING Customer
    ERRORFILES CustErr1, CustErr2;
```

```
DEFINE in_CustNum (INTEGER)
    ,in_Lname (CHAR(15))
    ,in_Fname (CHAR(10))
    ,in_Mailcode (INTEGER)
FILE=International.dat;
```

```
INSERT INTO Customer VALUES (
    :in_CustNum
    ,:in_Lname
    ,:in_Fname
    ,:in_Mailcode);
```

```
END LOADING;
LOGOFF;
```

END LOADING; indicates no more data and to start Phase 2.

FastLoad Fails to Complete

The output on the facing page provides an example of a FastLoad job that ran out of space in the database where the table was being loaded.

You can also see the list of each 100,000 rows as they are encountered.

The FastLoad job is shown below:

```
LOGON u4455/tfact01,tfact01;
BEGIN LOADING DS.Sales ERRORFILES DS.sales_1, DS.sales_2 INDICATORS;
DEFINE FILE=sales.dat;
INSERT INTO DS.Sales.*;
END LOADING;
LOGOFF;
```

The table definition is:

```
CREATE SET TABLE DS.Sales, NO FALBACK,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL
(
  store_id INTEGER NOT NULL,
  item_id INTEGER NOT NULL,
  sales_date DATE FORMAT 'YYYY-MM-DD',
  total_revenue DECIMAL(9,2),
  total_sold INTEGER,
  note VARCHAR(256) CHARACTER SET LATIN NOT CASESPECIFIC)
UNIQUE PRIMARY INDEX ( store_id, item_id, sales_date );
```



FastLoad Fails to Complete

```
0001 LOGON u4455/tfact01, ;
:
**** 09:42:20 Number of FastLoad sessions connected = 8
**** 09:42:20 FDL4808 LOGON successful
:
0002 BEGIN LOADING DS.Sales ERRORFILES DS.sales_err1, DS.sales_err2 INDICATORS;
:
0003 DEFINE FILE=sales.dat;
**** 09:42:27 FDL4803 DEFINE statement processed
:
0004 INSERT INTO DS.Sales.*;
:
**** 09:42:27 Number of recs/msg: 228
**** 09:42:27 Starting to send to RDBMS with record 1
**** 09:42:28 Starting row 100000
**** 09:42:28 Starting row 200000
**** 09:42:28 RDBMS error 2644: No more room in database DS.
**** 09:42:28 Increase database size and restart FastLoad
**** 09:42:28 Logging off all sessions
:
**** 09:42:33 Total processor time used = '1.26667 Seconds'
.
.   Start: Tue Mar 14 09:42:17 2006
.   End : Tue Mar 14 09:42:33 2006
.   Highest return code encountered = '12'.
**** 09:42:33 FastLoad Paused
```

Restarting FastLoad (Output)

Prior to restarting the job shown on the facing page, you must give the database more space to accommodate the load.

In the output example, you can see that:

- The restart log and error tables remain.
- The Loader repositions itself at the last checkpoint so that it can pick up loading from that point.
- After completing the restart, error tables are dropped and loading is complete.



Restarting FastLoad (Output)

```
0001 LOGON u4455/tfact01, ;
:
**** 09:53:17 Number of FastLoad sessions connected = 8
:
0002 BEGIN LOADING DS.Sales ERRORFILES DS.sales_err1, DS.sales_err2 INDICATORS;
0003 DEFINE FILE=sales.dat;
:
0004 INSERT INTO DS.Sales.*;
:
**** 09:53:20 Starting row 200000
**** 09:53:21 Starting row 300000
**** 09:53:22 Starting row 400000
**** 09:53:22 Sending row 483350
**** 09:53:22 Finished sending rows to the RDBMS
0005 END LOADING;
**** 09:53:41 END LOADING COMPLETE
Total Records Read      = 483350
Total Error Table 1     = 0 ---- Table has been dropped
Total Error Table 2     = 0 ---- Table has been dropped
Total Inserts Applied   = 483350
Total Duplicate Rows    = 0
Start: Tue Mar 14 09:53:24 2006
End : Tue Mar 14 09:53:41 2006
Note: These times apply to Phase 2.
0006 LOGOFF;
```

Restarting FastLoad – Summary

You may occasionally need to restart the FastLoad utility after already having started to load the table.

On the facing page are four scenarios you might encounter that would cause you to restart FastLoad. The scenarios also provide the steps you should take to execute the restart process.



Restarting FastLoad – Summary

Condition 1: Abort in Phase 1 – data acquisition incomplete.

Solution: Resubmit the script. FastLoad will begin from record 1 or the first record past the last checkpoint.

Condition 2: Abort occurs in Phase 2 – data acquisition complete.

Solution: Submit only BEGIN and END LOADING statements; restarts Phase 2 only.

Condition 3: Normal end of Phase 1 (paused) – more data to acquire, thus there is no 'END LOADING' statement in script.

Solution: Resubmit the adjusted script with new data file name. FastLoad will be positioned to record 1 or the first record past the last checkpoint.

Condition 4: Normal end of Phase 1 (paused) – no more data to acquire, no 'END LOADING' statement was in the script.

Solution: Submit BEGIN and END LOADING statements; restarts Phase 2 only.

Additional FastLoad Commands

Some additional FastLoad commands include:

Use **AXSMOD** to specify an access module (e.g., Reel Librarian file) that provides data to the FastLoad utility on network-attached client systems.

Use SESSIONS *max min* to specify the number of sessions; placed before the LOGON.

(*max*=maximum number of sessions that will be logged on; the *max* specification must be greater than zero). If you specify a *max* value larger than the number of available AMPs, FastLoad limits the sessions to one per working AMP.

The default maximum is one session for each AMP. Using the asterisk as the *max* specification logs on for the maximum number of sessions — one for each AMP.

min is optional; the minimum number of sessions required to run the job. The *min* specification must be greater than zero; default is 1.

Using the asterisk as the *min* specification logs on for at least one session, but less than or equal to the max specification.

Use **ERRLIMIT** to control a runaway error condition, such as an incorrect definition of the input data. Specify the maximum number of error records you want to occur before the system issues an ERROR and terminates the load.

Use **TENACITY** to specify the number of hours FastLoad will try to establish a connection. Default is no tenacity. The statement must be placed before LOGON.

Use **SLEEP** to specify the number of minutes FastLoad waits before retrying a logon. Default is 6 minutes. The statement must be placed before LOGON.

Use **DELETE** when you must empty an existing table prior to loading. It must precede the BEGIN LOADING statement. (It must be removed from the script prior to a restart.) Specify the table name and use ALL to indicate that you want all rows deleted. The option requires DELETE access right or privilege.

Use **DROP TABLE** in conjunction with the **CREATE TABLE** command to remove an old table and reestablish it. Use **DROP TABLE** to remove old error tables. These commands must precede the BEGIN LOADING statement and must be removed before a restart. This option requires DROP access right or privilege.

Use **HELP TABLE** to automatically acquire a DEFINE statement when the input data is an exact map of the table that you are loading. The **HELP TABLE** uses the current Data Dictionary definitions to generate the DEFINE.

Use **DATEFORM** to specify the form of the DATE data type for the job. This option is placed before LOGON.



Additional FastLoad Commands

AXSMOD	<i>name</i> [" <i>init-string</i> "] ;
SESSIONS	<i>max</i> [<i>min</i>] ;
ERRLIMIT	<i>max rejected records</i> ;
TENACITY	<i>hours</i> ; (default is no TENACITY)
SLEEP	<i>minutes</i> ; (default is 6 minutes)
DELETE FROM	<i>tablename</i> [ALL] ;
DROP TABLE	<i>tablename</i> ;
HELP TABLE	<i>tablename</i> ;
NOTIFY	OFF LOW MEDIUM HIGH ... ;
DATEFORM	INTEGERDATE ANSI DATE ;
SET SESSION CHARSET	" <i>charsetname</i> " ;
SET RECORD	FORMATTED ; UNFORMATTED ; BINARY ; TEXT ; VARTEXT "c" ;

Ex. If an input file has exactly the same field definitions as a table, then ...

```
DEFINE FILE = Trans_data;
INSERT INTO Trans.*;
```

FastLoad with Additional Options

For the FastLoad job on the facing page, the script uses the redirect process to acquire the input script (<) and direct the printed output of the FastLoad job (>).

The example also contains several parameters (SESSIONS, TENACITY, and SLEEP) that must be specified before the LOGON statement. Another option, DATEFORM, if used, must be placed before the LOGON statement.

The ERRLIMIT parameter may be specified before or after the LOGON statement.



FastLoad with Additional Options

```
fastload < /home/job1.fld > /home/job1.out
```

```
SESSIONS 12 8;
TENACITY 4;
SLEEP 3;
LOGON educ2/bank,bkpasswd;
ERRLIMIT 1000;

BEGIN LOADING Customer
ERRORFILES Cust_Err1, Cust_Err2;
DEFINE    in_CustNum      (INTEGER)
          ,in_SocSec     (INTEGER)
          ,Filler        (CHAR(40))
          ,in_Lname      (CHAR(30))
          ,in_Fname      (CHAR(20))
FILE=custdata.dat;
INSERT INTO CUSTOMER VALUES (
          :in_CustNum
          ,:in_Lname
          ,:in_Fname
          ,:in_SocSec);
END LOADING;
LOGOFF;
```

→ Input script file name & output file (report) name.

→ These options must be specified before LOGON.

→ Maximum number of error records before terminating.

→ Start Phase 1.

→ Start Phase 2. If omitted, FastLoad will pause.

Invoking FastLoad

The facing page displays the commands you can use to invoke the FastLoad utility in batch mode. The parameters for each command are listed in the three-column table.



Invoking FastLoad

Network Attached Systems: fastload [PARAMETERS] < *scriptname* >*filename*

Channel-Attached MVS Systems: // EXEC TDSFAST FDLOPT= [PARAMETERS]

Channel-Attached VM Systems: EXEC FAST [PARAMETERS]

Channel Parameter	Network Parameter	Description
BUFSIZE=kb	-b <i>kb</i>	Specifies input buffer size; maximum is 63 KB (default)
CHARSET= <i>charsetname</i>	-c <i>charsetname</i>	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets
ERRLOG= <i>filename</i>	-e <i>filename</i>	Alternate file specification for error messages; produces a duplicate record.
INMODETYPE=SAS_C	N/A	Specifies that the job will use an INMOD routine written in SAS/C.
SLEEP= <i>minutes</i>	-s <i>minutes</i>	Number of minutes that FastLoad pauses before retrying a logon.
TENACITY= <i>hours</i>	-t <i>hours</i>	Number of hours that FastLoad will continue trying to logon when the maximum number of load jobs are already running.
	< <i>scriptname</i>	Name of file that contains FastLoad commands and SQL statements.
	> <i>filename</i>	Name of output file for FastLoad messages.

INMOD

An INMOD is an exit routine that can precondition data and pass it on to the loader. You can write INMODs to pre-screen the input data being provided to FastLoad.

The INMOD and FastLoad use a return code value to communicate with each other.

You can write INMODs as restartable routines so that they synchronize with the loader's checkpoints.

Use INMODs to perform unusual conversions of data, for example, adding a sequenced column to the data, or reading data from a non-standard input file format.

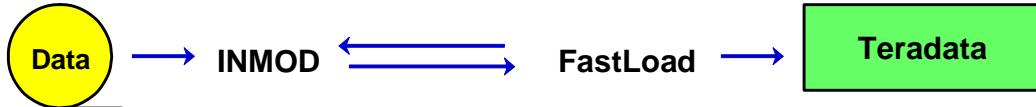
Note: When there is a serious problem, and the job must be terminated, the INMOD function must return a 4-byte integer.

To reference an INMOD routine, use the INMOD option with the DEFINE statement.

```
DEFINE     fieldname1    (INTEGER,  
                     fieldname2    (CHAR(4)),  
                      ...  
INMOD = name_of_inmod_routine;
```



INMODs



FastLoad requires a flat file as input. To acquire data from a non-standard data source, make unusual conversion of data, or otherwise condition the data, FastLoad can use an INMOD, or exit routine.

Communication needs to be established between the INMOD and FastLoad via return codes. The INMOD will pass data records to FastLoad.

INMOD to FastLoad return codes	<table border="1"> <tr> <td>0</td><td>Indicates that INMOD is returning a record in BODY.</td></tr> <tr> <td>Non 0</td><td>INMOD indicates end-of-file condition.</td></tr> </table>	0	Indicates that INMOD is returning a record in BODY.	Non 0	INMOD indicates end-of-file condition.				
0	Indicates that INMOD is returning a record in BODY.								
Non 0	INMOD indicates end-of-file condition.								
FastLoad to INMOD return codes	<table border="1"> <tr> <td>0</td><td>Calling for the first time. INMOD should open files to read data. FastLoad expects a record.</td></tr> <tr> <td>1</td><td>FastLoad expects a record.</td></tr> </table>	0	Calling for the first time. INMOD should open files to read data. FastLoad expects a record.	1	FastLoad expects a record.				
0	Calling for the first time. INMOD should open files to read data. FastLoad expects a record.								
1	FastLoad expects a record.								
FastLoad to INMOD additional return codes	<table border="1"> <tr> <td>2</td><td>FastLoad has been restarted. INMOD should position itself to the last checkpoint. FastLoad is not expecting a record and will not send a zero return code.</td></tr> <tr> <td>3</td><td>Indicates a checkpoint has been written and the INMOD should remember it. No record expected.</td></tr> <tr> <td>4</td><td>Indicates a Teradata failure; INMOD should position itself to the last checkpoint. No record expected.</td></tr> <tr> <td>5</td><td>FastLoad instructs the INMOD to clean up (workstations only).</td></tr> </table>	2	FastLoad has been restarted. INMOD should position itself to the last checkpoint. FastLoad is not expecting a record and will not send a zero return code.	3	Indicates a checkpoint has been written and the INMOD should remember it. No record expected.	4	Indicates a Teradata failure; INMOD should position itself to the last checkpoint. No record expected.	5	FastLoad instructs the INMOD to clean up (workstations only).
2	FastLoad has been restarted. INMOD should position itself to the last checkpoint. FastLoad is not expecting a record and will not send a zero return code.								
3	Indicates a checkpoint has been written and the INMOD should remember it. No record expected.								
4	Indicates a Teradata failure; INMOD should position itself to the last checkpoint. No record expected.								
5	FastLoad instructs the INMOD to clean up (workstations only).								

Application Utility Checklist

The facing page adds the FastLoad capabilities to the checklist.



Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED			
DML Functions	ALL	INSERT			
Multiple DML	Yes	No			
Multiple Tables	Yes	No			
Multiple Sessions	Yes	Yes			
Protocol Used	SQL	FASTLOAD			
Conditional Expressions	Yes	No			
Arithmetic Calculations	Yes	No			
Data Conversion	Yes	1 per column			
Error Tables	No	Yes			
Error Limits	No	Yes			
User-written Routines	No	Yes			
Support Environment (SE)	No	No			

Summary

The facing page summarizes some important concepts regarding the FastLoad utility.



Summary

FastLoad Features and Characteristics:

- Excellent utility for loading new or empty tables from a host or server.
- The empty table cannot have secondary indexes, join indexes, hash indexes, or Referential Integrity.
- Can reload previously emptied tables
 - Remove referential integrity or secondary indexes prior to using FastLoad.
- Full Restart capability
- Has two phases – creates an error table for each phase.
 - Error Limits and Error Tables, accessible using SQL

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Match the item in the first column to a corresponding statement in the second column.

- | | |
|---|--|
| 1. <input type="checkbox"/> Phase 1 | A. Might be used if a zero date causes an error |
| 2. <input type="checkbox"/> CHECKPOINT | B. Table status required for loading with FastLoad |
| 3. <input type="checkbox"/> ERRORTABLE1 | C. Records written in unsorted blocks |
| 4. <input type="checkbox"/> ERRORTABLE2 | D. Records rows with duplicate values for UPI |
| 5. <input type="checkbox"/> Empty Table | E. Not permitted on table to be loaded with FastLoad |
| 6. <input type="checkbox"/> Secondary Index | F. Points FastLoad to a record in an input file |
| 7. <input type="checkbox"/> Conversion | G. Can be used to restart loading from a given point |
| 8. <input type="checkbox"/> NULLIF | H. Records constraint violations |
| 9. <input type="checkbox"/> RECORD | I. Builds the actual table blocks for the new table |
| 10. <input type="checkbox"/> Phase 2 | J. Transform one data type to another, once per column |

Lab Exercise 34-1

The BTEQ syntax to create the two data files for this exercise is:

```
.LOGON      ...;  
  
.EXPORT DATA FILE = data34_1;  
EXEC AP.Lab34_1_1;  
.EXPORT RESET  
  
.EXPORT DATA FILE = data34_2;  
EXEC AP.Lab34_1_2;  
.EXPORT RESET  
  
.LOGOFF;
```

These macros have WHERE clauses that SELECT specific data rows. These macros limit the number of rows that are selected; therefore there is no need to include the LIMIT parameter with the BTEQ .EXPORT statement.

After creating these data files, you may want to check their size to ensure that you have created them correctly. An easy way to do this is to use the UNIX **ls -l** command.

The size of **data34_1** should be 244,000 bytes.

The size of **data34_2** should be 183,000 bytes.

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab34_12.fld** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function

To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 34-1

Purpose

In this lab, you will set up a restartable FastLoad operation.

What you need

You need to create two data input sets and use your empty Customer table. The input data sets will get their data from the AP.Customer table.

Tasks

1. Using two separate BTEQ EXPORT commands, create two source data sets, data34_1 and data34_2. The SQL for selecting the appropriate rows is contained in the macros AP.LAB34_1_1 (for data34_1) and AP.LAB34_1_2 (for data34_2).
Note: data34_1 has 4000 records and data34_2 has 3000 records
2. Create a FastLoad script that loads the first 4000 records (data34_1 file) and do not include the END LOADING statement in this script.
3. Create a FastLoad script that loads the additional 3000 records (data34_2) and complete the FastLoad.
4. Check the result. (Your Customer table should contain 7000 rows.)

Lab Exercise 34-2

The BTEQ syntax to create the data file for this exercise is:

```
.LOGON    ...;  
.EXPORT DATA FILE = data34_3;  
EXEC AP.Lab34_2;  
.EXPORT RESET  
.LOGOFF;
```

After creating this data file, you may want to check its size to ensure that you have created it correctly. An easy way to check the size in UNIX is to use the UNIX **ls -l** command.

The size of **data34_3** should be 495,000 bytes.

Note: FastLoad requires that DECIMAL be spelled out (DEC causes a syntax error).

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab34_22.fld** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function
To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 34-2

Purpose

In this lab, you will set up a FastLoad operation that takes incoming dates that have a value of 0 (actually 0000-00-00) and converts them to NULL.

What you need

An empty TRANS table and the macro AP.Lab34_2.

Tasks

1. Use BTEQ EXPORT and the macro AP.Lab34_2 to create a source data file (data34_3). This macro outputs the DATE in character format and the year is output as 4 characters.
2. FastLoad the data from the file data34_3 to your empty TRANS table. In this exercise, the default format for a date is character (10) with a format of YYYY-MM-DD. The data file has dates set to 0 (zero) that must be converted to NULL.

(Hint: Use a FORMAT 'YYYY-MM-DD' on the INSERT and a NULLIF='0000-00-00' on the DEFINE. You must define incoming DATE as a character field.)
3. How many rows in your table have a NULL Trans_Date? _____

Teradata Training

Notes

Module 35



The Support Environment for FastExport, MultiLoad, and TPump

After completing this module, you will be able to:

- Explain the elements of the Support Environment.
- Use the Support Environment to invoke a utility.
- Process parameter input from a host file.
- Use system and user-defined variables.
- Write messages to an output file.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Support Environment	35-4
Setting Up the Support Environment	35-6
Invoking Utilities	35-8
Support Environment Commands	35-10
Initializing the Log Table	35-12
Initialization and Wrap Up Commands	35-14
.ACCEPT – Working with Variables.....	35-16
Support Environment System Variables	35-18
.DISPLAY and .ROUTE Commands.....	35-20
Example: Using Variables in a Script	35-22
Working with Control Logic	35-24
Support Environment Example – Input	35-26
Support Environment Example – Output.....	35-28
Teradata SQL Support	35-30
Script – Example Input	35-32
Script – Example Output.....	35-34
Summary	35-36
Review Questions	35-38
Lab Exercise 35-1	35-40

Support Environment

To ensure consistency for application utilities, Teradata provides the Support Environment, a sophisticated utility platform that makes *fully automatic* restarts available. Without needing to know the reason for the failure and without needing to change the script, you can restart a job by resubmitting the script.

The Support Environment supports a complete range of SQL commands (except SELECT), and permits conditional processing of SQL and utility commands with an easy-to-use .IF, .THEN, .ELSE, and .ENDIF facility.



Support Environment

- Provides a common environment – language, functions, flexibility, etc. – for utilities such as MultiLoad, FastExport, and TPump.
- Provides a fully-nested .RUN file facility.
- Interprets utility commands and provides error reporting.
- Supports a wide range of DDL and DML commands.
- Allows for conditional processing of commands.
- Supports system-defined and user-defined variables.
- Allows logic to be applied both before and after the utility executes.
- Provides recovery management from a Teradata or host failure.

Setting Up the Support Environment

Utilities are not called by the user directly, but are invoked by the Support Environment after initial housekeeping tasks are complete.

The setup process requires the naming of a restart log table, which then governs the operation completely.

You can follow the setup commands with SQL statements for preparing the job.



Setting Up the Support Environment

- Utilities are invoked by the Support Environment after the preparatory statements have been executed:

.LOGTABLE
.LOGON

- These may be followed by SQL statements to:
 - Set the default database.
 - Establish a CHECKPOINT in the Permanent Journal.
 - CREATE and DROP Tables, etc.
 - Set system variables.
 - Retrieve control parameters from outside sources (database or flat file).
- Comments may be included with scripts.

/* This job is used to*/

Invoking Utilities

The facing page identifies the commands you can use to invoke the utilities that use the Support Environment.

If from its interrogation of the restart log, the Support Environment determines the present operation to be a restart, the Support Environment accepts responsibility for not submitting a previously successful operation a second time.

The facing page also shows an example of job code that calls the Support Environment.

Note: The script on the facing page runs the FastExport utility. FastExport is discussed in more detail in a later module.



Invoking Utilities

- **MultiLoad** is invoked with: .BEGIN [IMPORT] MLOAD
or
.BEGIN DELETE MLOAD
- **FastExport** is invoked with: .BEGIN EXPORT
- **TPump** is invoked with: .BEGIN LOAD
- Upon initialization, the Support Environment reviews the Restart Log Table to determine whether or not this is a restart.
- If it is, the Support Environment will not submit a previously successful statement a second time.

Support Environment Commands

```
.LOGTABLE CustLog_fxp ;  
.LOGON .....;  
  
.BEGIN EXPORT;  
.EXPORT OUTFILE Cust_file;  
SELECT * FROM Customer;  
.END EXPORT;  
  
.LOGOFF ;
```

Support Environment Commands

Note that *all* Support Environment and utility commands are preceded with a period (.). Any statement NOT preceded by a period is presumed to be SQL and is sent to the Teradata database for processing.



Support Environment Commands

.LOGTABLE	Acquires or creates the Restart Log Table.
.LOGON	Connects multiple sessions to Teradata.
.LOGOFF	Terminates the utility operation.
.DATEFORM	Specify INTEGERDATE or ANSIDATE.
.ACCEPT	Input parameters to Support Environment.
.RUN	Specifies an external script file.
.IF ... THEN [.ELSE]	Identifies statements to be executed if certain conditions are true or false
.ENDIF	Required to terminate a .IF condition.
.DISPLAY	Writes messages to a specific destination.
.ROUTE	Specifies output file other than SYSPRINT or standard output.
.SET	Assigns a data type and value to a variable.
.SYSTEM	Submits an operating system command to the client environment.

Initializing the Log Table

Utilities use information in the Restart Log Table to restart jobs halted because of a Teradata or client system failure.

If a utility completes with a return code of zero, the Restart Log Table is automatically dropped.



Initializing the Log Table

- The **.LOGTABLE** command is required.
- **.LOGTABLE** and **.LOGON** commands must be the first statements processed (either directly or via the **.RUN** command).
- **.LOGTABLE creates a new table or acquires an existing Log Table.**
- Privileges required for the Log Table:
 - CREATE TABLE (to create a new table)
 - INSERT
 - UPDATE
 - SELECT
- By default, the log table is created in your default database which requires PERM space.
- The format of the Log Table is unique to each of the utilities (FastExport, MultiLoad, and TPump).

Initialization and Wrap Up Commands

The facing page displays initialization and wrap up commands for the Support Environment.

Utilities deliver to the Host a “high watermark” return code. If this value is zero, all work tables, empty error tables, and the restart log table are dropped. Any return code of 8 or greater indicates an aborted job.



Initialization and Wrap Up Commands

.LOGON [*tdpid /*] *username , [password [, 'acctid']] ;* *Utility will run any startup string defined for the user.*

.LOGOFF [*retcode*] ; *Permits user to specify return-code; otherwise, high watermark is returned.*

.RUN FILE *fileid*
[IGNORE
 { *charpos1*
charpos 1 THRU charpos2
THRU charpos2
charpos1 THRU charpos2 }] ;
Identifies external source of control statements and optionally ignores extraneous data - nest up to 16 levels of .RUN.

When the Utility terminates, it returns to the HOST with a “high watermark” return code:

- 00 Successful completion
- 04 Warning
- 08 User error
- 12 Severe internal error
- 16 No output message available

.ACCEPT – Working with Variables

The ACCEPT command can...

- Accept from a single data record from an external source, and use it to set one or more utility variables.
- Accept from a system variable and use it to set a utility variable.

You can treat input values for the Support Environment in whole or in part. The IGNORE function of the .ACCEPT statement permits the ACCEPTed data to contain filler data. For example, you can also use the IGNORE to ignore sequence numbers in the first 6 columns.

When ACCEPTing from a **fileid**, the **fileid** can be any of the following:

- with VM, a FILEDEF name
- with MVS, a DDNAME
- with UNIX and Windows, a pathname for a file
- an * which represents the system console or standard input (*stdin*) device.

Multiple values in the input record are space separated. Character values must be delimited with single quotes. For example,

'Los Angeles'	90210	is valid
Los Angeles	90210	is not valid

If the input file contains multiple records, the ACCEPT command will only accept from the first record.

If the number of variables is greater than the number of values in the input record, then unused variables are undefined or NULL.

If the number of values in the input record is greater than the number of variables, you will receive a warning message.



.ACCEPT – Working with Variables

The ACCEPT command can ...

- accept from a single data record from an external source, and use it to set one or more utility variables.
- accept from a system variable and use it to set a utility variable.

```
.ACCEPT var, . . . [FROM] FILE fileid  
[ IGNORE  
  { charpos1  
    charpos1 THRU  
    THRU charpos2  
    charpos1 THRU charpos2 } ] ;
```

```
.ACCEPT var [FROM] ENVIRONMENT VARIABLE env_var ;  
ENV VAR
```

Utility variables will be replaced by their values before text is displayed.

For display or evaluation, variable names must be preceded with an ampersand (&).

If the value is a character, the variable name must be enclosed in single quotes.

Support Environment System Variables

The facing page displays some of the variables you can use within the Support Environment. See the reference manuals for the complete set of variables.

The .SET command has to precede the .BEGIN EXPORT (MLOAD or LOAD) command.

Note: The Return Code is the return code from the last Teradata Database command.

Miscellaneous notes on Time and Date variables:

- &SYSDATE – returns 8-character date in *yy/mm/dd* format
- &SYSDATE4 – returns 10-character date in *yyyy/mm/dd* format
- &SYSDAY – returns 3-character uppercase day of week specification: MON, TUE, WED, THU, FRI, SAT or SUN
- &SYSTIME – returns 8-character time in *hh:mm:ss* format

For these 4 variables, the original values are maintained after a utility restart operation.

Note that because the values are all character data types, you should not reference them in numeric operations.



Support Environment System Variables

.SET var [TO] expression; *Permits a variable to be set or reset to an expression or a pre-existing variable.*

Example: .SET dbase TO 'tfact30';
 .SET tname TO 'customer';

<u>System Variables</u>	<u>Description</u>	<u>Format</u>	<u>Example</u>
&SYSDATE	System Date	YY/MM/DD	07/10/22
&SYSDATE4	System Date	YYYY/MM/DD	2007/10/22
&SYSTIME	System Time	HH:MM:SS	16:11:33
&SYSDAY	Day of Week	X(3)	MON
&SYSOS	Host Op System	X(5)	UNIX or Win32
&SYSUSER	MVS/UNIX User Id		uljc30
&SYSRC	Return Code		0

MultiLoad Specific Variables (not complete list)

&SYSDELCNT_	Delete Count	Ex., &SYSDELCNT1, &SYSDELCNT2, ...
&SYSINSCNT_	Insert Count	Ex., &SYSINSCNT1, &SYSINSCNT2, ...
&SYSUPDCNT_	Update Count	
&SYSETCNT_	Error Table Count	
&SYSUVCNT_	Uniqueness Violation Count	
&SYSRCDCNT_	Count of import records read	
&SYSRJCTCNT_	Count of records rejected from import file	Note: n is 1 to 5

.DISPLAY and .ROUTE Commands

When DISPLAYing to a **fileid**, the **fileid** can be any of the following:

- with VM, a FILEDEF name
- with MVS, a DDNAME
- with UNIX and Windows, a pathname for a file
- an * which represents the system console or standard output (*stdout*) device.

In UNIX, **/dev/tty** references the user's terminal device directly. **/dev/tty** is not the same as standard output.

The DISPLAY command creates a new file or replaces an existing file; it does **not** append to an existing file. However, multiple DISPLAY commands to the same filename in the same script are all placed into the same file.

The ECHO function on the .ROUTE command permits messages to be sent to multiple destinations.



.DISPLAY and .ROUTE Commands

.DISPLAY 'text' [TO] FILE fileid; *Used to write messages to specified fileid.*

.ROUTE MESSAGES [TO] FILE fileid *Changes routing of default output.*

[[WITH] ECHO [TO] FILE fileid] ; *'ECHO' permits routing to default and a second copy anywhere in script.*

Examples:

.DISPLAY 'Run Date - &SYSDATE4' TO FILE /dev/tty;

Run Date - 2007/10/22

.ACCEPT home FROM ENV VAR HOME;

.DISPLAY 'The \$HOME directory is &home' TO FILE *;

The \$HOME directory is /home/tfact30

(* - the output is directed to standard output device.)

.ROUTE MESSAGES TO FILE /tmp/mldrun1.out WITH ECHO TO FILE /dev/tty;

Example: Using Variables in a Script

The facing page contains an example of using variables in a FastExport Script.

Notes:

- The .SYSTEM command is the UNIX remove file command with the –f or force option. The –f option removes the file without prompting the user.
- The script on the facing page runs the FastExport utility. FastExport is discussed in more detail in a later module.
- The two periods (..) between the &dbase and &tname are needed to represent a single period. If a single period was used, the support environment would interpret the text immediately after the single period as a command.
- If an ampersand is needed in the script, use &&.



Example: Using Variables in a Script

Example:

```
.LOGTABLE CustLog_fxp ;
.LOGON .....;
.ACCEPT home FROM ENV VAR HOME;
.SET dbase      TO 'tfact30';
.SET tname       TO 'customer';
.SET expname    TO '&home/cust_file';
.SYSTEM 'rm -f &expname';
.DISPLAY 'Exporting data file &expname'
TO FILE /dev/tty;
.BEGIN EXPORT;
.EXPORT OUTFILE &expname;
SELECT * FROM &dbase..&tname;
.END EXPORT;
.LOGOFF ;
```

Output:

Data file saved in UNIX:
`/home/tfact30/cust_file`

Output to terminal screen:
`Exporting data file /home/tfact30/cust_file`

Portion of FastExport output:

```
0011 SELECT * FROM &dbase..&tname;
**** 16:23:10 UTY2402 Previous statement
modified to:
0012 SELECT * FROM tfact30.customer;
```

Working with Control Logic

The facing page describes the use of .IF, .ELSE, and .ENDIF statements to apply conditional logic to your job.

The *conditional expression* is an expression that can be evaluated as either true or false.

When evaluation of the expression returns a numeric result:

- Zero is interpreted as false
- Nonzero results are interpreted as true

The Support Environment utilities (MultiLoad, FastExport, and TPump) support the nesting of .IF commands up to 100 levels.



Working with Control Logic

.IF conditional expression THEN ;

*if condition is true,
then execute these statement(s) ;*

.IF is followed by a conditional expression that initiates execution of subsequent commands and statements.

[.ELSE ;]

*if condition is false,
then execute these statement(s) ;*

.ELSE is followed by commands and statements which execute when the preceding IF command is false.

.ENDIF ;

.ENDIF delimits the group of commands and statements subject to previous IF or ELSE commands.

Note:

The Support Environment utilities (MultiLoad, FastExport, and TPump) support the nesting of .IF commands up to 100 levels.

Support Environment Example – Input

The example on the facing page demonstrates a number of the features of the Support Environment, including:

- The .RUN facility
- The .IF/.ENDIF function
- Using system variables
- Displaying messages to an output file
- Initializing MultiLoad (MLOAD)

Note: The Support Environment is case-sensitive for variables and input data.



Support Environment Example – Input

```
.LOGTABLE CustLog_mld ;  
  
.RUN FILE /home/kac/logon ;  
  
.IF '&SYSDAY' NE 'FRI' THEN ;  
  .DISPLAY 'This job runs on Friday'  
    TO FILE /tmp/display_out ;  
  .LOGOFF ;  
.ENDIF ;  
  
.BEGIN IMPORT MLOAD  
...  
...
```

Create or Acquire Restart Log Table.

Run commands in file logon.

Check Day of Week. Write a message and terminate Job if not 'FRI' (Case-specific).

Invoke utility.

/home/kac/logon

```
.LOGON dev4450/KAC,amber96;
```

Support Environment Example – Output

The Support Environment performs a *preliminary* syntax check of all *utility* statements prior to calling the utility. It also resolves all variables and writes messages to output files as directed. The resolutions are *not* dynamic. Once a variable has been resolved, it remains resolved across application restarts.

Thus, if **&SYSDAY** has once been resolved to 'FRI', and the job later aborts, upon restart, **&SYSDAY** remains resolved to 'FRI' even though the actual day of the week may have changed.



Support Environment Example – Output

MultiLoad Utility Output

Logon / connection

```
0001 .LOGTABLE CustLog_mld ;
0002 .RUN FILE /home/kac/logon;
0003 .LOGON dev4450/KAC, ;
13:28:42 FRI OCT 19, 2007
UTY6211 A successful connect was made to the DBC.
13:28:43 FRI OCT 19, 2007
UTY6211 Logtable 'KAC.Cust_Logtable' has been created.
```

Processing Control Statements

```
0004 .IF '&SYSDAY' NE 'FRI' THEN ;
13:28:43 FRI OCT 19, 2007
UTY2402 Previous statement modified to:
0005 .IF 'FRI' NE 'FRI' THEN;
0006 .DISPLAY 'This job runs on Friday'
TO FILE /tmp/display_out;
0007 .LOGOFF;
0008 .ENDIF;
0009 .BEGIN IMPORT MLOAD
```

Teradata SQL Support

The Support Environment supports a full range of SQL functionality, except for SELECT.

Note:

Specifying any DML statements (INSERT /UPDATE/DELETE) before specifying the utility BEGIN command (e.g., BEGIN MLOAD) will use the non-fast path and processing will be done as normal SQL statements and the Transient Journal will be used as needed. This may be very slow depending on the SQL statement. Specifying the DML statement after the utility BEGIN command (e.g., BEGIN MLOAD) will use the fast path. For example, in MultiLoad, the processing will be done in the utility transaction phase which is very fast.



Teradata SQL Support

- The Support Environment supports:
 - Utility operations.
 - DML and DDL functions for preparatory tasks in the same job-step, avoiding multiple utility invocations.
- Examples of TERADATA SQL statements that can be used include:

CREATE DATABASE	ALTER TABLE
MODIFY DATABASE	DROP TABLE, VIEW, MACRO, INDEX
DELETE DATABASE	CREATE TABLE, VIEW, MACRO, INDEX
DROP DATABASE	REPLACE VIEW, MACRO
	RENAME TABLE, VIEW, MACRO
DATABASE	
CHECKPOINT	INSERT
COLLECT STATISTICS	UPDATE
COMMENT ON	DELETE
SET SESSION COLLATION	GRANT
	REVOKE
RELEASE MLOAD	GIVE

Note: User-generated transactions (BT, ET) and SELECT are not supported.

Script – Example Input

Multiple input variables from a file treated by the .ACCEPT command are separated by a space. Text values must be enclosed in single quotes.

The Control_Table has the following columns:

ID	Status
2	Text



Script – Example Input

Host File: **datain**

0	0	0	2		'	T	e	x	t	'
---	---	---	---	--	---	---	---	---	---	---

The script is for setting up a MultiLoad operation. The database table is named Control_Table. It has columns named Status and ID.

```
.LOGTABLE restartlog2_mld;
.LOGON Ist4900/tfact05,password ;
.ACCEPT num, name FROM FILE datain ;
.SET nbrin to 2 ;
.IF &num = &nbrin THEN;
    UPDATE Control_Table SET Status = '&name' WHERE ID = &num ;
    DISPLAY 'Update of record &nbrin successful'
        TO FILE /tmp/display2_out ;
.ENDIF ;
.LOGOFF ;
```

Script – Example Output

Notice how the output shows the resolution of these values before the utility is called.



Script – Example Output

Host File: datain

0	0	0	2	'	T	e	x	t	'
---	---	---	---	---	---	---	---	---	---

MultiLoad Utility

```

10:15:08 Processing start date FRI MAR 17, 2006
Logon/connection
0001 .LOGTABLE Restartlog2_mld ;
0002 .LOGON lst4900/tf05, ;
UTY6211 A successful connect was made to the DBC
UTY6217 Logtable 'TF05.Restartlog2_mld' has been created.
Processing Control Statements
0003 .ACCEPT num, name FROM FILE datain;
0004 .SET nbrin to 2 ;
0005 .IF &num = &nbrin THEN;
UTY2402 Previous statement modified to:
0006 .IF 2=2 THEN;
0007 UPDATE Control_Table SET Status = '&name' WHERE ID = &num ;
UTY2402 Previous statement modified to:
0008 UPDATE Control_Table SET Status = 'Text' WHERE ID = 2 ;
UTY1016 'UPDATE' request successful
0009 .DISPLAY 'Update of record &nbrin successful' TO FILE /tmp/display2_out;
UTY2402 Previous statement modified to:
0010 .DISPLAY 'Update of record 2 successful' TO FILE /tmp/display2_out;
0011 .ENDIF ;
0012 .LOGOFF ;

```

Summary

The facing page summarizes some of the important concepts regarding the Support Environment.



Summary

Support Environment:

- Common environment for utilities such as MultiLoad, FastExport, and TPump.
- Provides error reporting.
- Supports a wide range of DDL and DML commands for one-step jobs.
- Allows for conditional processing.
- Supports system- and user-defined variables.
- Provides recovery management from a Teradata or host failure.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Match the item in the first column to its corresponding statement in the second column.

- | | |
|---------------------------------------|---|
| <input type="checkbox"/> 1. .LOGTABLE | A. Connects sessions to Teradata |
| <input type="checkbox"/> 2. .LOGON | B. Uses a single data record to set one or more utility variables |
| <input type="checkbox"/> 3. .ACCEPT | C. System variable |
| <input type="checkbox"/> 4. UPDATE | D. Identifies the log to create or acquire |
| <input type="checkbox"/> 5. &SYSDATE | E. Teradata SQL statement permitted by Support Environment |

Lab Exercise 35-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.



Lab Exercises

Lab Exercise 35-1 (Optional)

Purpose

In this lab, you will set up the Support Environment, and take data from an input record (data35_1) and insert it to a row in your customer table.

What you need

AP.Customer and your Customer table.

Tasks

1. Create a file or data set (data35_1) and enter the following data for INSERT into the Customer table:

Customer_Number	10001
Last_Name	'YourLastName'
First_Name	'YourFirstName'
Social_security	333445555

Use the format:

10001 'YourLastName' 'YourFirstName' 333445555 (items separated by spaces)

2. Prepare a Support Environment script that defines the record to the Support Environment, using the ACCEPT to read the record and use the SET command to dynamically modify the table name in your INSERT statement. Use FastExport to execute this script.
3. Test the result: `SELECT * FROM Customer WHERE Customer_Number = 10001;`

Teradata Training

Notes

Module 36



FastExport

After completing this module, you will be able to:

- State FastExport capabilities.
- Describe how sorted output is produced from a multiple-session SELECT.
- Prepare a FastExport script.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

FastExport	36-4
.BEGIN and .END EXPORT	36-6
SESSIONS <i>max min</i>	36-6
TENACITY and SLEEP	36-6
.END EXPORT	36-6
.EXPORT	36-8
A FastExport Script.....	36-10
The SELECT Request.....	36-12
Impact of Requesting Sorted Output.....	36-14
The SORT Procedure	36-16
Multiple Exports in one FastExport Job	36-18
Invoking FastExport.....	36-20
FastExport and Variable Input	36-22
Selection Controls	36-22
A FastExport Script with ACCEPT	36-24
A FastExport Script with LAYOUT	36-26
.LAYOUT, .FIELD, and .FILLER.....	36-28
INMODs and OUTMODs.....	36-30
OUTMOD Return Codes	36-32
Application Utility Checklist	36-34
Summary	36-36
Review Questions	36-38
Lab Exercise 36-1	36-40
Lab Exercise 36-2	36-42

FastExport

FastExport is designed to outperform BTEQ .EXPORT in the transfer of large amounts of data from the larger Teradata database systems to the host using multiple sessions.

FastExport is NOT designed to make the Teradata Database perform faster. It is designed to make greater use of multiple Parsing Engines and AMPs as well as multiple channels.



FastExport

- Exports large volumes of formatted data from Teradata to a host file or user-written application.
- Takes advantage of multiple sessions.
- Export from multiple tables.
- Uses Support Environment.
- Fully automated restart.
- Uses one of the “Loader” slots.



Teradata
Database

.BEGIN and .END EXPORT

The BEGIN EXPORT command signifies the beginning of an export task and sets the specifications for the task sessions with the Teradata Database.

SESSIONS *max min*

max is maximum number of FastExport sessions that will be logged on when you enter a LOGON.

- The *max* specification must be greater than zero.
- If you specify a SESSIONS *max* value larger than the number of available AMPs, the utility limits the sessions to one per working AMP.
- The default maximum is 4 for UNIX and networked systems.
- Using the asterisk character as the *max* specification logs on for the maximum number of sessions — one for each AMP.
- *min* is optional, the minimum number of sessions required to run the job. The *min* specification must be greater than zero. The default minimum, if you do not use the SESSIONS option or specify a min value, is 1.
- Using the asterisk character as the *min* specification logs on for at least one session, but less than or equal to the max specification.

SESSIONS * * has the effect of not using the SESSIONS parameter at all.

TENACITY and SLEEP

Tenacity specifies the number of hours that the FastExport utility tries to log on to the Teradata Database.

When the FastExport utility tries to log on for a new task, and the Teradata Database indicates that the maximum number of utility import/export sessions are already running, the FastExport utility:

1. Waits for six minutes, by default, or for the amount of time specified by the SLEEP option.
2. Then it tries to log on to the Teradata Database again.
3. The FastExport utility repeats this process (steps 1 and 2) until it has either logged on for the required number of sessions or exceeded the TENACITY *hours* time period.

.END EXPORT

The .END EXPORT command indicates that the Support Environment has completed its syntax check and housekeeping activities and instructs FastExport to send the SELECT(s) to the Teradata database.



.BEGIN and .END EXPORT

.BEGIN EXPORT

SESSIONS	max min
TENACITY	hours
SLEEP	minutes
NOTIFY	<u>OFF</u> LOW MEDIUM HIGH ... ;

SESSIONS

- Maximum, and optionally, minimum number of sessions the utility may use - defaults to 4 for UNIX FastExport.
- The utility will log on two *additional* SQL sessions: one for the Restart Log and one for the SELECT.

TENACITY

- Number of hours FastExport will try to establish a connection to the system; default is 4.

SLEEP

- Number of minutes that FastExport will wait between logon attempts; default is 6.

NOTIFY

- Parameter for specifying the notify user exit option
- The FastExport manual specifies in detail which events are associated with each level.

.END EXPORT;

- Delimits a series of commands that define a single EXPORT action.
- Causes the utility to send the SELECT(s) to the Teradata Database.

.EXPORT

The .EXPORT statement permits the definition of the output data file and optionally an AXSMOD and/or OUTMOD routine. The reference manual contains the details on using the AXSMOD and OUTMOD options.

Only RECORD and INDICATOR output modes are permitted, since FastExport performance relies heavily on large amounts of data being returned (maybe millions of rows) and this is considered unsuitable for generation of reports. Therefore, there is no FIELD mode with FastExport. INDICATOR is the default.

The BLOCKSIZE parameter defaults to 63.5 KB.

The FORMAT option only applies to UNIX and Windows systems. The options for FORMAT are:

FASTLOAD	a two-byte integer, n , followed by n bytes of data, followed by an end-of-record marker, either X ‘0A’ or X ‘0D’.
BINARY	a two-byte integer, n , followed by n bytes of data
TEXT	an arbitrary number of bytes followed by an end-of-record marker, either X ‘0A’ or X ‘0D0A’ for Windows systems.
UNFORMAT	exported as it is received from CLI without any client modifications.

FORMAT	Length Indicator (2 bytes)	End-of-Record Marker (Hex ‘0A’ – 1 byte)
FastLoad	Y	Y
Binary	Y	N
Text	N	Y
Unformat	N	N

Note: The Length Indicator does not include itself or the Hex ‘0A’. Indicator bytes follow the Length Indicator or they are at the start of the record.

The MLSRIPT option causes FastExport to generate a MultiLoad script that can be used to load the exported data back into Teradata.



.EXPORT

```
.EXPORT OUTFILE fileid [ AXSMOD name [ 'init-string' ] ] [ OUTMOD module_name ]  
[ MODE      RECORD | INDICATOR ]  
[ BLOCKSIZE integer ]  
[ FORMAT    FASTLOAD | BINARY | TEXT | UNFORMAT ]  
[ OUTLIMIT  record_count ]  
[ MLSRIPT   fileid ] ;
```

- MODE** If RECORD, then indicator bytes for NULLs are not included in exported data.
If INDICATOR, then indicator bytes for NULLs are included in exported data.
- BLOCKSIZE** Defines the maximum block size to be used in returning exported data.
Default (and maximum) is 63.5 KB.
- FORMAT** Record format of the export file on network-attached UNIX and Windows platforms.
- OUTLIMIT** Defines the maximum number of records to be written to the output host file.
- MLSRIPT** FastExport generates a MultiLoad script that can be used later to load the exported data back into a Teradata system.

A FastExport Script

FastExport is called from the Support Environment using the initialization procedure. FastExport requires a Restart Log Table that must be identified with the .LOGTABLE statement.



A FastExport Script

Define Restart Log



```
.LOGTABLE RestartLog1_fxp;
```

Specify number of sessions



```
.RUN      FILE logon ;
.SET     CityName TO 'Los Angeles';
.SET     ZipCode TO 90066;
```

Destination file



```
.EXPORT  OUTFILE custacct_data;
```

Via a SELECT, specify the columns and rows to export.



```
SELECT   A.Account_Number
        , C.Last_Name
        , C.First_Name
        , A.Balance_Current
FROM    Accounts A           INNER JOIN
        Accounts_Customer AC  INNER JOIN
        Customer C
ON      C.Customer_Number = AC.Customer_Number
ON      A.Account_Number = AC.Account_Number
WHERE   A.City      = '&CityName'
AND     A.Zip_Code  = &ZipCode
ORDER BY 1 ;
```

Send request.



```
.END EXPORT ;
```

Terminate sessions



```
.LOGOFF ;
```

The SELECT Request

FIELD MODE (formatted output for reports), is not available with FastExport. The WITH and WITH BY operators, which provide sub-totals and grand totals, are not supported.

FastExport permits multiple statement SELECTs.



The SELECT Request

- Defines the data to be returned to the host, server, or client workstation.
- The job may consist of multiple SELECT statements which will be executed sequentially by FastExport.
- Applies normal transaction locks (READ lock) which are fully automatic.
- These locks are normally held by the utility until all response rows have been moved to AMP spool, and then are released.
- Supports the “LOCKING ROW (or TABLE *tablename*) FOR ACCESS” modifier to request an “access lock”.
- Restrictions – you cannot use SELECT (in a FastExport job) with the following:
 - Non-data tables (e.g. CURRENT_DATE, ...)
 - Equality condition for a Primary Index or USI
 - WITH option to generate total or subtotal response rows.
 - The USING modifier to submit data parameters as a constraint to the SELECT.

Impact of Requesting Sorted Output

The facing page describes the sort process.

Sorting exported data adds additional overhead to the FastExport job. Only sort the exported data rows if it is necessary.



Impact of Requesting Sorted Output

A special FastExport “sort protocol” is used to take advantage of multiple sessions. Each session transfers data a block at a time from multiple AMPs.

This protocol includes the following steps:

- The SELECT request is fully processed in the normal way using DBC/SQL protocol.
- At this point, response data is maintained in spool, sorted locally by the AMPs.
- Two further distributions between the AMPs (using the BYNET) are required to complete the sort.

Sort notes:

- Requesting sorted data adds additional work (overhead and time) to Teradata.
- If the exported rows are to be loaded back into a Teradata DB (e.g., MultiLoad), there probably is no need to sort the exported rows.

The SORT Procedure

Response rows are initially placed in the AMP local spool and sorted. They are then redistributed over the BYNET in such a way that all values from the first sort value are placed on the first logical AMP (which is randomly selected); values from the second sort value are placed on the next physical AMP and so on, round-robin until all data is sorted.

This process is known as the VERTICAL distribution.

HORIZONTAL distribution takes place as blocks of data are built taking all values for the first sort value from the first AMP, all values for the second sort value from the next AMP and so on, round-robin until the block is full.

Multiple sessions are then used to return the sorted blocks in sequence to the host.

Unlike a normal data sort, this procedure is comparatively resource-intensive. It is counter-balanced by the improved performance possible, with multiple sessions and block transfer to the host.

The SORT Procedure

Response rows locally sorted in SPOOL:

AMP 1

ADAMS
BOYCE
FIELD
JONES
SMITH
WILSON

AMP 2

BATES
DAVIS
KIEL
NICHOLS
PETERS
TIBBS
TOMS

AMP 3

BOYCE
CHARLES
HERBERT
POTTER

AMP 4

ADAMS
DAVIS
GEORGE
HANCOCK
HERBERT
MERCER

Vertical Distribution:

ADAMS
ADAMS
DAVIS
DAVIS
HERBERT
HERBERT
NICHOLS
TIBBS

BATES
FIELD
JONES
PETERS
TOMS

BOYCE
BOYCE
GEORGE
KIEL
POTTER
WILSON

CHARLES
HANCOCK
MERCER
SMITH

Horizontal Distribution:

BLOCK 1
ADAMS
ADAMS
BATES
BOYCE
BLOCK 5
NICHOLS
PETERS
POTTER
SMITH

BLOCK 2
BOYCE
CHARLES
DAVIS
DAVIS
BLOCK 6
TIBBS
TOMS
WILSON

BLOCK 3
FIELD
GEORGE
HANCOCK
HERBERT

BLOCK 4
HERBERT
JONES
KIEL
MERCER

Multiple Exports in one FastExport Job

A FastExport job can contain multiple .BEGIN EXPORT; and .END EXPORT; pairs as shown on the facing page.



Multiple Exports in one FastExport Job

cust_trans.fxp

```
.LOGTABLE RestartLog2_fxp ;
.LOGON ..... ;
.DISPLAY 'Exporting Cust_file - &SYSDATE4' TO FILE /dev/tty;
.BEGIN EXPORT;
.EXPORT OUTFILE Cust_file;
    SELECT * FROM Customer_1;
    SELECT * FROM Customer_2;
.END EXPORT;
.DISPLAY 'Exporting Trans_file - &SYSDATE4' TO FILE /dev/tty;
.BEGIN EXPORT;
.EXPORT OUTFILE Trans_file;
    SELECT * FROM Transactions;
.END EXPORT;
.LOGOFF ;
```

To execute: fexp < cust_trans.fxp > cust_trans.out

Exported data file: **Cust_file** Output to screen: **Exporting Cust_file - 2007/10/22**
Exported data file: **Trans_file** Output to screen: **Exporting Trans_file - 2007/10/22**

Invoking FastExport

The facing page displays the commands you can use to invoke the FastExport utility in batch mode. The parameters for each command are listed in the three-column table.

If you want to use FastExport in interactive mode, enter the command **fexp** at your system command prompt.

Optionally, when FastExport starts, it can read a configuration file to establish defaults for the FastExport job. On network-attached systems (e.g., UNIX and Windows), MultiLoad can read configuration parameters from a file named **fexpcfg.dat**. This file is located either in the current directory or the directory referenced by the variable FEXPLIB.

On channel-attached systems, the DD statement for the MultiLoad configuration file must be labeled FEXPCFG.

There are 7 parameters you can set in this file.

- CHARSET=character-set-name
- ERRLOG=filename
- BRIEF=on/off
- MAXSESS=max-sessions
- MINSESS=min-sessions
- STATUS=ON/OFF
- DATAENCRYPTION=ON/OFF

The values that you specify in the FastExport configuration file override the internal utility default values for these parameters. Configuration file parameters can be overridden with runtime parameters. The order of preference (highest to lowest) for these parameters is:

- 1 – Runtime parameters
- 2 – FastExport script parameters
- 3 – Configuration file parameters
- 4 – FastExport default values

The FastExport utility automatically checks for a configuration file each time you invoke the utility. Upon locating a configuration file, the utility sets the defaults as specified, produces the appropriate output messages and begins processing your FastExport job.

If the configuration file cannot be opened, or if the FastExport utility encounters syntax errors in the file, the utility produces an output message, disregards the error condition and begins processing your FastExport job. An invalid configuration file entry does not abort your FastExport job.

If there is no configuration file, the utility begins processing your FastExport job without an error indication. The configuration file is an optional feature of the FastExport utility, and its absence is not considered to be an error condition.



Invoking FastExport

Network Attached Systems: `fexp [PARAMETERS] < scriptname >outfilename`

Channel-Attached MVS Systems: `// EXEC TDSFEXP FEXPPARM= [PARAMETERS]`

Channel-Attached VM Systems: `EXEC FASTEXPT [PARAMETERS]`

Channel Parameter	Network Parameter	Description
BRIEF	-b	Reduces print output runtime to the least information required to determine success or failure.
CHARSET= <i>charsetname</i>	-c <i>charsetname</i>	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets.
ERRLOG= <i>filename</i>	-e <i>filename</i>	Alternate file specification for error messages; produces a duplicate record.
" <i>fastexport command</i> "	-r ' <i>fastexport cmd</i> '	Signifies the start of a FastExport job; usually a RUN FILE command that specifies the script file.
MAXSESS= <i>max sessions</i>	-M <i>max sessions</i>	Maximum number of FastExport sessions logged on.
MINSESS= <i>min sessions</i>	-N <i>min sessions</i>	Minimum number of FastExport sessions logged on.
	< <i>scriptname</i>	Name of file that contains FastExport commands and SQL statements.
	> <i>outfilename</i>	Name of output file for FastExport messages.

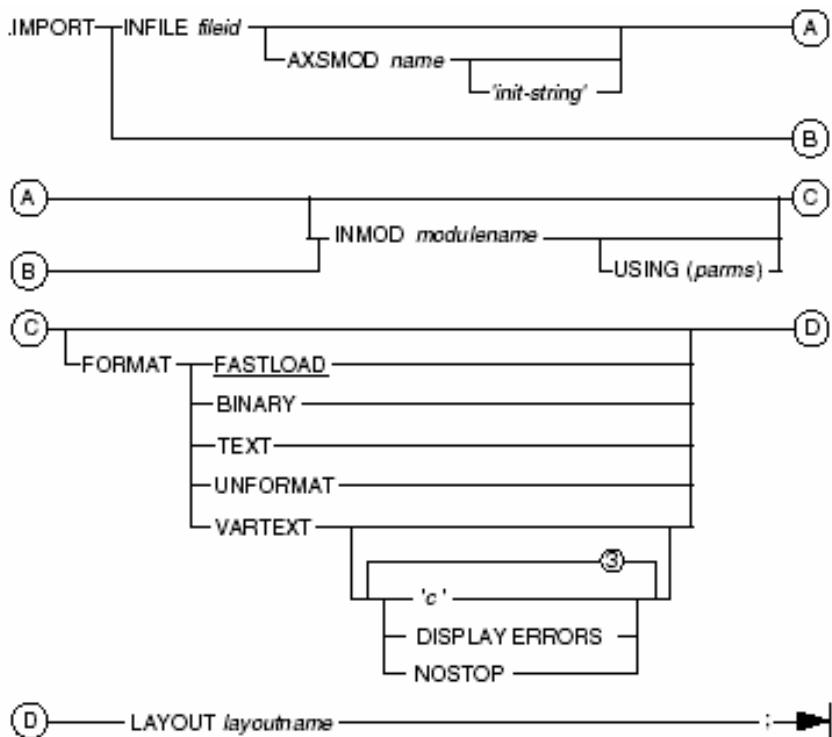
FastExport and Variable Input

The facing pages provide additional information regarding FastExport. Variable input to FastExport can either be accepted from a parameter file or you can IMPORT from a data file.

Selection Controls

This page defines the optional use of an IMPORT data set. Data from this data set can be used as dynamic variables in the SELECT statement(s) of the FastExport.

This allows you to run the same SELECT statement multiple times, each with a different dynamic value. The results of all of the occurrences of the SELECT statement are sent to a single FastExport data set.



The FORMAT options are similar to those described earlier for .EXPORT. VARTEXT specifies that each record is variable length text record format, with each field separated by a delimiter character.



FastExport and Variable Input

Selection Controls

- There are two techniques that can be used to provide variable input to FastExport.
 - ACCEPT from a parameter file; only accept from a single record.
 - IMPORT from a data file; each import record is applied to every SELECT.
- Read input variables from a host input data file described by the .LAYOUT command.
- Apply *each* input variable value to *every* SELECT in the exact order listed in the FastExport script before reading the next.
- Defines a host file as the source of the data values required for the SELECT REQUEST.
- Permits the use of a user-written INMOD routine to (optionally) read and (always) process the input record before passing it to the utility.

A FastExport Script with ACCEPT

The script shown on the facing page adds an ACCEPT command to the script shown earlier.



A FastExport Script with ACCEPT

parmfile1

'Los Angeles' 90066

par_City par_Zip

ACCEPT variables from input record.

Reference accepted variables with an &.

```
.LOGTABLE RestartLog1_fxp;
.RUN      FILE logon ;
.ACCEPT   par_City, par_Zip FROM FILE parmfile1;
.BEGIN    EXPORT SESSIONS 4 ;
.EXPORT   OUTFILE custacct_data;
SELECT    A.Account_Number
          , C.Last_Name
          , C.First_Name
          , A.Balance_Current
FROM      Accounts A           INNER JOIN
          Accounts_Customer AC  INNER JOIN
          Customer C
ON        C.Customer_Number = AC.Customer_Number
ON        A.Account_Number = AC.Account_Number
WHERE     A.City      = '&par_City'
AND       A.Zip_Code  = &par_Zip
ORDER BY  1 ;
.END EXPORT ;
.LOGOFF   ;
```

A FastExport Script with LAYOUT

The script shown on the facing page adds LAYOUT commands to the script shown earlier.



A FastExport Script with LAYOUT

city_zip_infile

Los Angeles	90066
San Diego	90217

in_City

in_Zip

IMPORT fields from input records.

Reference imported fields with a :

```
.LOGTABLE RestartLog1_fxp;
.RUN      FILE logon ;
.BEGIN    EXPORT SESSIONS 4 ;
.LAYOUT   Record_Layout ;
.FIELD    in_City          1 CHAR(20) ;
.FIELD    in_Zip           * CHAR(5) ;
.IMPORT   INFILE city_zip_infile LAYOUT Record_Layout ;
.EXPORT   OUTFILE cust_acct_outfile2 ;
SELECT    A.Account_Number
          ,C.Last_Name
          ,C.First_Name
          ,A.Balance_Current
FROM      Accounts A
INNER JOIN Accounts_Customer AC
INNER JOIN Customer C
ON        C.Customer_Number = AC.Customer_Number
ON        A.Account_Number = AC.Account_Number
WHERE     A.City      = :in_City
AND       A.Zip_Code  = :in_Zip
ORDER BY  1 ;
.END EXPORT ;
.LOGOFF ;
```

.LAYOUT, .FIELD, and .FILLER

These commands permit the user to concatenate input data both vertically and horizontally, and to reduce the size of the input data parcel by converting fixed-length input fields to variable.



.LAYOUT, .FIELD and .FILLER

```
.LAYOUT layoutname
  [ CONTINUEIF position = variable ]
  [ INDICATORS ] ;
```

.LAYOUT

- Describes the layout of externally stored records used to supply values for SELECT.
- Followed by .FIELD and .FILLER commands.

```
.FIELD fieldname { startpos datadesc } || fieldexp [ NULLIF nullexpr ]
  [ DROP {LEADING / TRAILING } { BLANKS / NULLS }
  [ [ AND ] {TRAILING / LEADING } { NULLS / BLANKS } ] ] ;
.FILLER [ fieldname ] startpos datadesc ;
```

.FIELD

- Input fields supporting redefinition and concatenation.

Startpos identifies the start of a field relative to 1.

Fieldexpr specifies a concatenation of fields in the format: fn1 || fn2 [|| fn3 ...]

The option **DROP LEADING / TRAILING BLANKS / NULLS** is applicable only to character datatypes, and is sent as a VARCHAR with a 2-byte length field.

.FILLER

- Identifies data NOT to be sent to the Teradata database.

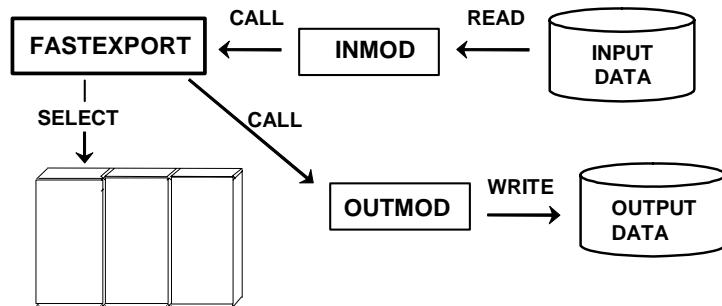
INMODs and OUTMODs

In addition to supporting INMODs for processing input data, FastExport permits the use of OUTMODs for processing data returned to the host.

This feature provides you with the ability to use a single step to process the returned data in any way necessary for the application.

The IMPORT portion of FastExport will be covered after the initial discussion of MultiLoad.

INMODs and OUTMODs



INMODs

- Read input data values from a file.
- Qualify **SELECT** requests.
- Usually more applicable to an import utility such as MultiLoad.

OUTMODs

- Process answer-set data.
- Modify, discard, or record responses.
- Usually more applicable to an export utility such as FastExport.

OUTMOD Return Codes

Return codes sent to the OUTMOD from FastExport are similar in function to those used for INMODs, but involve data flowing in the other direction — from Teradata to the host.

Details on return code 5 are:

Teradata Database Restart Entry — specifies the first call that resumes processing after a Teradata Database restart.

If the FastExport utility was writing to an output file when the Teradata Database failed, then the utility repositions that file before calling the OUTMOD routine.

This call signifies that the OUTMOD routine should also reposition its data or files as needed and execute checkpoint procedures to resume processing.

If the Teradata Database restarts before the first checkpoint, entry code 2 is sent for cleanup, and the entry code 1 is sent to re-initialize the job.

Details on return code 6 are:

Client Restart Entry – Specifies the first call that resumes processing after a client system restart.

If the FastExport utility was writing to an output file when the client system failed, then the utility repositions that file before calling the OUTMOD routine. This call signifies that the OUTMOD routine should also reposition its data or files as needed and execute checkpoint procedures to resume processing

If the client system restarts before the first checkpoint, entry code 1 is sent to re-initialize the job.



OUTMOD Return Codes

FastExport OUTMOD Return Codes

1	<p>Initial Entry — Specifies the initial entry call that the FastExport utility makes before sending the first SELECT statement to the Teradata Database.</p> <p>There is only one initial entry call for a FastExport job. If the utility is in a restart mode, the utility uses an entry code value of 5 or 6 to specify the first call after a Teradata Database or client system restart.</p>
2	<p>End of Response Entry — Specifies the end of response call that the Fast Export utility makes after receiving the last row of export data from the Teradata Database.</p>
3	<p>Response Row Entry — Specifies a response row call that the FastExport utility makes for each row of export data from the Teradata Database.</p>
4	<p>Checkpoint Entry — Specifies a checkpoint call that the FastExport utility makes after processing the last response row for each SELECT statement.</p> <p>This call signifies that the OUTMOD routine should capture checkpoint data to support a restart operation if the Teradata Database or client system fails.</p>
5	<p>Teradata Database Restart Entry — Specifies the first call that resumes processing after a Teradata Database restart.</p>
6	<p>Client Restart Entry — Specifies the first call that resumes processing after a client system restart.</p>

Application Utility Checklist

The facing page adds the FastExport capabilities to the checklist.



Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED	Yes (SE)		
DML Functions	ALL	INSERT	SELECT		
Multiple DML	Yes	No	Yes		
Multiple Tables	Yes	No	Yes		
Multiple Sessions	Yes	Yes	Yes		
Protocol Used	SQL	FASTLOAD	EXPORT		
Conditional Expressions	Yes	No	Yes		
Arithmetic Calculations	Yes	No	Yes		
Data Conversion	Yes	1 per column	Yes		
Error Files	No	Yes	No		
Error Limits	No	Yes	No		
User-written Routines	No	Yes	Yes		
Support Environment (SE)	No	No	Yes		

Summary

Remember that FastExport is not designed to make the Teradata database perform faster. It *is* designed to take full advantage of multiple Parsing Engines, mainframe channels, and the LAN.

The facing page summarizes some important concepts regarding the FastExport utility.



Summary

- Best choice for exporting large amounts of data from the Teradata database to a host file using multiple sessions.
- Fully automatic restart capability.
- Specialized processing of output data can be handled using an OUTMOD routine.
- Starting with Teradata V2R6.1, the MaxLoadTasks and MaxLoadAWT parameters determine the maximum number of utility jobs that can execute at a given time.
 - Teradata can accommodate not more than a combined total of 60 utility jobs at any one time (FastLoad, MultiLoad, FastExport).
 - Up to 30 of these can be FastLoad, MultiLoad.
 - The FastExport limit is 60 minus the number of active FastLoad and MultiLoad jobs. (FastLoad, MultiLoad, FastExport).

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Answer True or False.

1. **True or False.** FastExport requires the use of a PI or USI in the SELECTs.
2. **True or False.** The number of FastExport sessions (for a UNIX server) defaults to the number of AMPs.
3. **True or False.** The maximum block size you can specify with FastExport is 128 KB.
4. **True or False.** You can export from multiple tables with FastExport.
5. **True or False.** You can use multiple SELECTs in one FastExport job.
6. **True or False.** The default lock for a SELECT in a FastExport job is a table level ACCESS lock.

Lab Exercise 36-1

The facing page describes the tasks of this lab exercise.

The size of the data file (data36_1) should be 1,005,000 bytes.

With UNIX systems, to view the output text from FastExport on your screen and place the output text into a file, you can use the following option:

fexp < scriptname.fxp | tee scriptname.out

To append the text to a file, use the –a (append option)

fexp < scriptname.fxp | tee –a scriptname.out

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab36_11.fxp** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function

To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 36-1

Purpose

In this lab, you will use FastExport to create an export file that contains one record for each transaction. You will have to join columns from two different tables in order to create the export file.

What you need

Populated AP.Accounts and AP.Trans tables.

Tasks

1. Create a FastExport script that outputs to file data36_1. For each transaction in the AP.Trans table, include the transaction_number, account_number, number, street, city, state, and the zipcode of the associated account (AP.Accounts).
2. Run the script.
3. Test the result by using the UNIX ls -l command.

Lab Exercise 36-2

The facing page describes the tasks of this lab exercise.

The exported file should have 439 rows.

Note: The following UNIX command will provide the number of rows in the output report.

```
wc -l report36_2
```

Output should look like:

Account_Number	City	Balance_Current	Above MAX or Below MIN
-----------------------	-------------	------------------------	-------------------------------

SQL Hints:

The CASE can be used to generate the Below MIN or Above MAX literal for the output.

```
SELECT      Account_Number      (CHAR(12)),  

          City           (CHAR(12)),  

          Balance_Current   (FORMAT '$$$$$,$$9.99') (CHAR(12)),  

          (CASE  

              WHEN Balance_Current < &LoVal THEN 'Below MIN'  

              WHEN Balance_Current > &HiVal THEN 'Above MAX'  

          END)  

FROM        AP.Accounts  

WHERE       ...
```

Note: CAST can also be used to format the output from the SELECT.

```
SELECT  

    CAST (Account_Number AS CHAR(12)),  

    CAST (City AS CHAR(12)),  

    CAST (CAST (Balance_Current AS FORMAT '$$$$$,$$9.99') AS CHAR(12)),  

    ...
```

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab36_22.fxp** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function
To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 36-2

Purpose

In this lab, you will use FastExport to read input data from a data set / file or accepting input values as a parameter, and export a report to another data set / file. In order to produce readable output, all selected data should be converted to FIXED CHARACTER as outlined below:

What you need

Populated AP.Accounts table.

Tasks

This exercise involves the FastExport of 'EXCEPTION' data, a list of Accounts which either fall below a minimum Balance_Current or exceed a maximum value and are from the cities in the input data set. The output file (report) mode should be RECORD and format should be TEXT.

1. Create an input file named data36_2 with 1 line of input: '**Los Angeles**'
2. Prepare a FastExport script which does the following:
 - a. Treats this as a parameter file and ACCEPT from it. Treat this data as variable input for the SELECT.
 - b. Uses the .SET command to initialize two variables: LoVal 500 and HiVal 9499
 - c. Includes a SELECT statement that projects ACCOUNT NUMBER, CITY, BALANCE CURRENT, and a character string of either BELOW MIN or ABOVE MAX and sorts by Account_Number. Simply display 'BELOW MIN' or 'ABOVE MAX' as a literal with the SELECT. Use CHAR to convert the Account_Number from INTEGER to CHAR data.
 - d. Creates an output file named **report36_2**. Note: MODE RECORD and FORMAT TEXT
3. Run the test and view the result using the UNIX *more* command.

Teradata Training

Notes

Module 37



MultiLoad

After completing this module, you will be able to:

- **Describe the capabilities of MultiLoad.**
- **Name the five phases of MultiLoad and state the main function of each.**
- **Create a MultiLoad script.**
- **Run a script to update/load table(s) using MultiLoad.**
- **Explain the advantages of using MultiLoad.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

What is MultiLoad?	37-4
MultiLoad Limitations	37-6
How MultiLoad Works	37-8
Advantages of MultiLoad	37-10
Basic MultiLoad Statements (for Import Tasks).....	37-12
Sample MultiLoad IMPORT Task.....	37-14
IMPORT Task.....	37-16
5 Phases of IMPORT Task.....	37-18
Phase 1: Preliminary	37-20
Phase 2: DML Transaction	37-22
Phase 3: Acquisition.....	37-24
Phase 3: Acquisition – a Closer Look	37-26
Phase 4: Application	37-28
Phase 4: Application – a Closer Look.....	37-30
Phase 5: Cleanup.....	37-32
Execute END MLOAD processing as an explicit transaction	37-32
MLOAD Session Logoff.....	37-32
Sample MultiLoad DELETE Tasks	37-34
DELETE Task Differences from IMPORT Task.....	37-36
A Closer Look at DELETE Task Application Phase	37-38
MultiLoad Locks.....	37-40
Utility locks.....	37-40
Restarting MultiLoad	37-42
RELEASE MLOAD Statement	37-44
Invoking MultiLoad	37-46
Application Utility Checklist	37-48
Summary	37-50
Review Questions	37-52
Lab Exercise 37-1	37-54

What is MultiLoad?

MultiLoad is a batch mode utility that runs on the host system. It is used for loading, updating or deleting data to and from populated tables, typically with batch inputs from a host file.

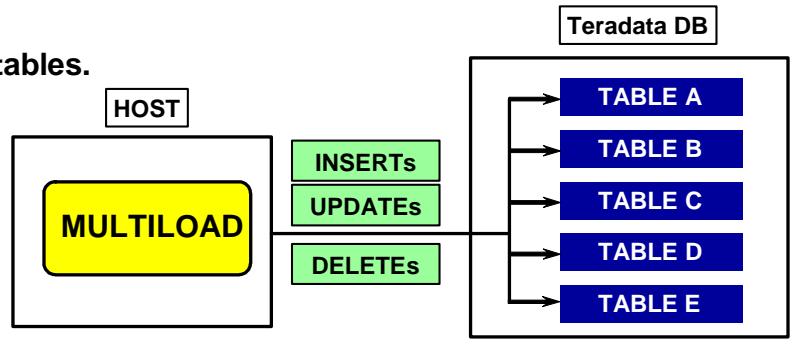
MultiLoad has many features that make it appealing for maintaining large tables:

- Uses FastLoad-like technology to accomplish TPump-like functionality.
- Support for up to five tables per import task.
- Tables may contain pre-existing data, but cannot have Unique Secondary Indexes nor can it have Referential Integrity.
- Ability to perform multiple maintenance operations with one pass of input data files.
- Ability to perform conditional maintenance based on 'apply' condition.
- Ability to do INSERTs, UPDATEs, DELETEs and UPSERTs (UPDATE if exists, else INSERT).
- Each affected data block is written only once.
- Host and LAN support.
- Full Restart capability using a Log file, even with AMPs down.
- Programmable error limits.
- Error reporting via error files.
- Support for INMODs to customize data being loaded—although less likely.

The DBSCtrl parameter MaxLoadTasks defines the maximum number of utilities (FastLoad, FastExport, and MultiLoad) that can run on the system at one time.

What is MultiLoad?

- Batch mode utility that runs on the host system.
- FastLoad-like technology – TPump-like functionality.
- Supports up to five populated tables.
- Multiple operations with one pass of input files.
- Conditional logic for applying changes.
- Supports INSERTs, UPDATEs, DELETEs and UPSERTs; typically with batch inputs from a host file.
- Affected data blocks only written once.
- Host and LAN support.
- Full Restart capability.
- Error reporting via error tables.
- Support for INMODs.



MultiLoad Limitations

MultiLoad is a very powerful and flexible utility. Some **MultiLoad** restrictions are:

- No data retrieval capability (i.e., no **SELECT** statement).
- Concatenation of input data files is not allowed.
- Host (APPLY clause) will not process arithmetic functions (i.e., ABS, LOG, etc.).
- Host will not process exponentiation or aggregate operators (i.e., AVG, SUM, etc.).
- Cannot process tables with Unique Secondary Indexes (USIs) defined.
- Import tasks require use of Primary Index.

If any of the above limitations are significant to your ability to load a table, you might want to consider alternatives:

- Write an INMOD for use with MultiLoad.
- Use TPump.
- Use FastLoad.



MultiLoad Limitations

- No data retrieval capability.
- Concatenation of input data files is not allowed.
- Host will not process arithmetic functions.
- Host will not process exponentiation or aggregates.
- Cannot process tables defined with USI's, Referential Integrity, Join Indexes, Hash Indexes, or Triggers.
 - Soft Referential Integrity is supported
- Import tasks require use of Primary Index.

Alternatives:

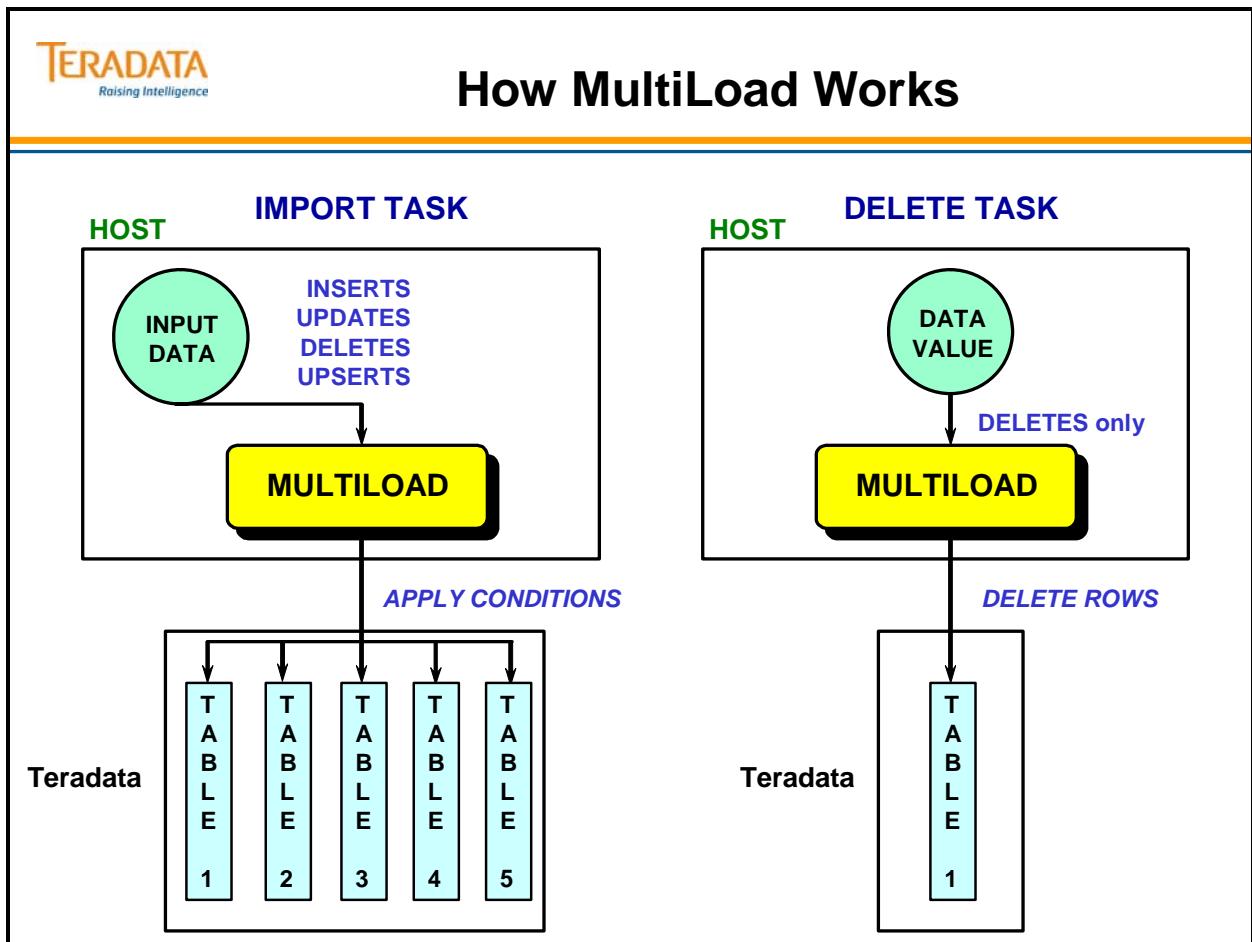
- Write an INMOD for use with MultiLoad.
- Use TPump.
- Use FastLoad.

How MultiLoad Works

MultiLoad typically uses an input file that is read to run batch-like maintenance actions against data on the Teradata database. It allows **INSERT**, **DELETE**, **UPDATE** and **UPSERT** operations against up to five tables per import task.

There are two distinct types of tasks that MultiLoad can perform:

- | | |
|--------------------|--|
| IMPORT task | Intermix a number of different SQL/DML statements and apply them to up to five different tables depending on the APPLY conditions. |
| DELETE task | Execute a single DELETE statement on a single table, often with a single value as a condition for the deletion (i.e., <code>DELETE FROM TABLE_1 WHERE COL_A > 921000;</code>). This value may be hard-coded or supplied from a host file. |



Advantages of MultiLoad

The advantages of MultiLoad are listed on the facing page.



Advantages of MultiLoad

- Minimizes the use of the PEs.
- Gets input data to the AMPs as quickly as possible.
- Uses multiple-AMP sessions.
- **Uses the parallelism of the AMPs to apply changes.**
- **Keeps BYNET activity low with AMP-local processing.**
- **Avoids Transient Journaling overhead.**
- Allows Checkpoint/Restartability even with down AMPs.
- Prevents lengthy rollbacks of aborted jobs.
- Allows for maximum access to table during processing.
- Posts errors to special error tables.
- Provides extensive processing statistics.

Basic MultiLoad Statements (for Import Tasks)

The following is an explanation of common components of a MultiLoad IMPORT script:

.LOGTABLE defines the table name of the Restart Log.

.LOGON defines username, which will own the sessions.

.BEGIN MLOAD TABLES defines the tables, which will participate in the MultiLoad.

.LAYOUT defines the layout of the incoming record(s).

.FIELD defines the name of an input field, its position in the record, and its data type.
(Absolute positioning may be done with a number, or relative positioning with an asterisk,
“*”.)

.FILLER defines input data that will not be sent to the database table. A **.FILLER** statement allows a name and requires a starting position or asterisk, and the data type.

.DML LABEL defines a set of DML instructions, which will be applied if conditions are met.

.IMPORT INFILE references the name of the input file.

FROM m FOR n optionally defines starting # and ending #
THRU k of records to process from input file.

FORMAT options – FASTLOAD
 BINARY
 TEXT
 UNFORMAT
 VARTEXT ‘’

LAYOUT references previously defined **LAYOUT**.

APPLY references **LABEL** to be applied and conditions under which to do so.

.END MLOAD; defines end of MultiLoad script.

.LOGOFF; terminate the sessions.



Basic MultiLoad Statements (for Import Tasks)

```
.LOGTABLE [ restartlog_tablename ] ;
.LOGON [ tdpid/userid, password ] ;
.BEGIN MLOAD TABLES [ tablename1, ... ] ;
.LAYOUT [ layout_name ] ;
.FIELD ..... ;
.FILLER ..... ;
.DML LABEL [ label ] ;
.IMPORT INFILE [ filename ]
    [ FROM m ] [ FOR n ] [ THRU k ]
    [ FORMAT FASTLOAD | BINARY | TEXT | UNFORMAT | VARTEXT 'c' ]
    LAYOUT [ layout_name ]
    APPLY [ label ] [ WHERE condition ] ;
.END MLOAD ;
.LOGOFF ;
```

```
.FIELD fieldname { startpos datadesc } || fieldexp [ NULLIF nullexpr ]
    [ DROP {LEADING / TRAILING } { BLANKS / NULLS }
    [ [ AND ] {TRAILING / LEADING } { NULLS / BLANKS } ] ] ;

.FILLER [ fieldname ] startpos datadesc ;
```

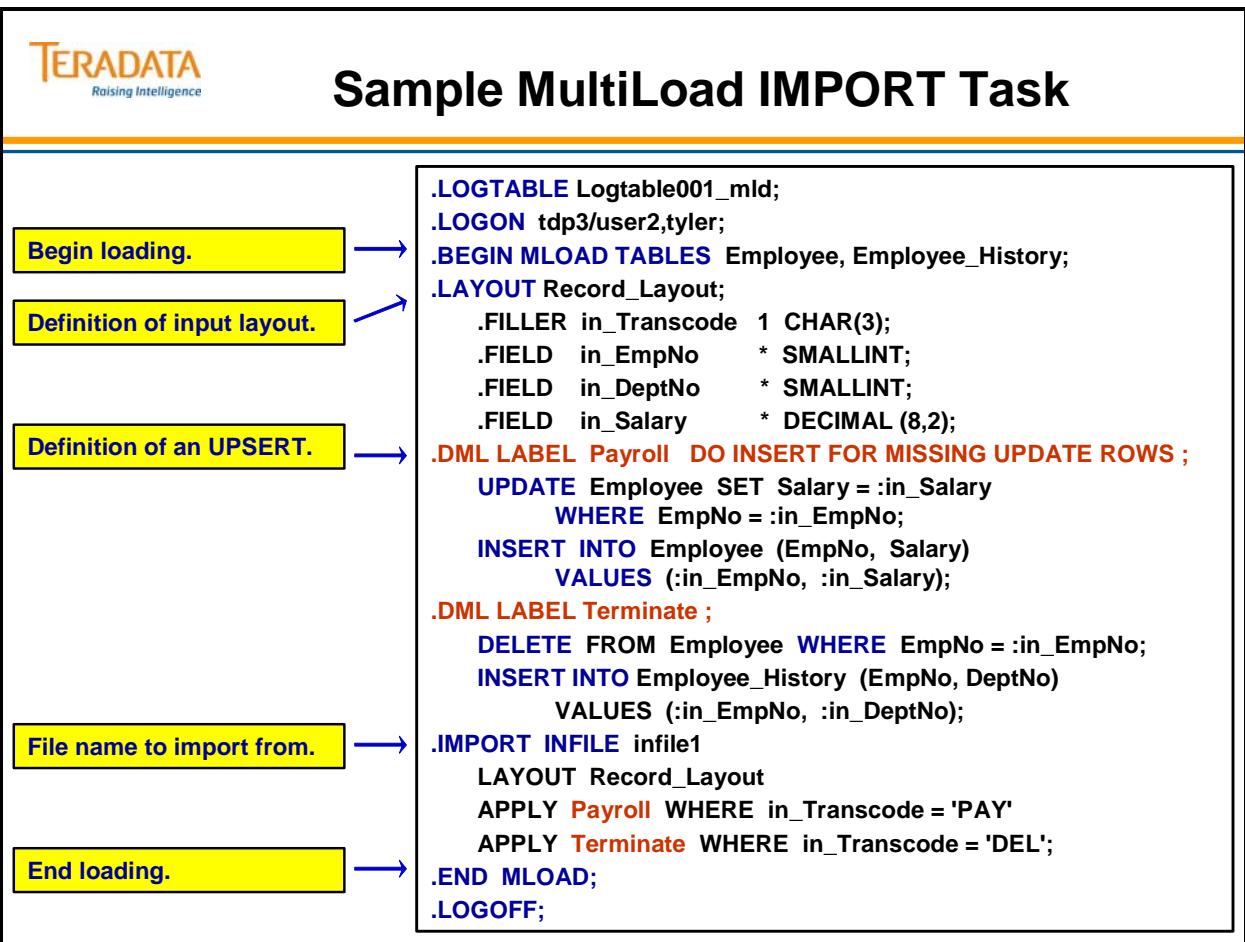
Sample MultiLoad IMPORT Task

The script on the facing page updates, inserts, and deletes, depending on the conditions for the employee table.

Each import task can include multiple INSERT, UPDATE, and DELETE statements, and the multiple DML operations can be conditionally applied to as many as five tables with a single pass of the client file.

The key words “**DO INSERT FOR MISSING UPDATE ROWS**” indicate an UPSERT operation. An UPSERT requires consecutive UPDATE and INSERT statements following the .DML LABEL statement.

If the UPDATE statement fails because the target table row does not exist, MultiLoad automatically executes the INSERT statement, completing the operation in a single pass instead of two.



IMPORT Task

IMPORT tasks are used to do multiple combinations of **INSERTs**, **DELETEs**, **UPDATEs** and **UPSERTs** to one or up to five tables. Updates that change the value of a table's primary index are not permitted. You may change the value of a column based on its current value (i.e., $\text{COL} = \text{COL} + 10$).

IMPORT tasks cannot be done on tables with Unique Secondary Indexes.



IMPORT Task

- INSERTs, DELETEs, UPDATEs and UPSERTs allowed.
- Up to a maximum of five tables:
 - Empty or populated.
 - NUSIs permitted.
- MultiLoad Import task operations are always primary index operations - however, you are not allowed to change the value of a table's primary index.
- Change the value of a column based on its current value.
- Permits non-exclusive access to target tables from other users except during Application Phase.
- Input error limits may be specified as a number or percentage.
- Allows restart and checkpoint during each operating phase.
- IMPORT tasks cannot be done on tables with USI's, Referential Integrity, Join Indexes, Hash Indexes, or Triggers.
 - With V2R5, IMPORT tasks can be done on tables defined with “Soft Referential Integrity”.

5 Phases of IMPORT Task

IMPORT consists of five separate phases of processing. They are:

Preliminary phase Basic setup

DML phase Get DML steps down on AMPS

Acquisition phase Send the input data to the AMPS and sort it

Application phase Apply the input data to the appropriate target tables

End phase Basic clean up



5 Phases of IMPORT Task

Preliminary

Basic set up

DML Transaction

Send the DML steps to the AMPs

Acquisition

Send the input data to the AMPs

Application

Apply the input data to appropriate table(s)

Cleanup

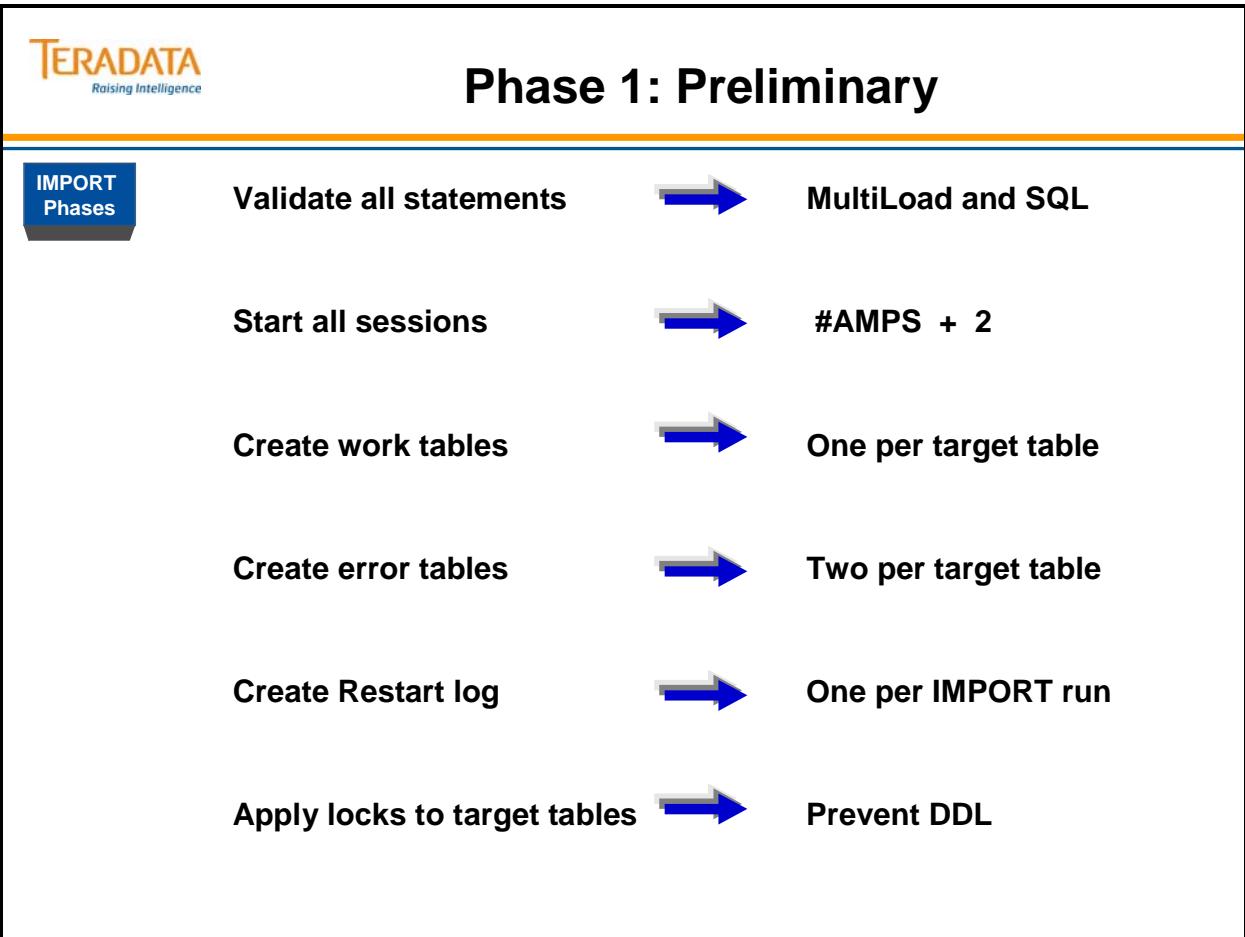
Basic clean up

Details

Phase 1: Preliminary

The first of IMPORT's five phases is the **Preliminary**. It performs the following tasks:

Validate all statements	All MultiLoad and SQL statements are validated and syntax checked.
Start all sessions	Typically, one MLOAD session per AMP plus two control sessions. One is for handling the SQL and logging and the second is an alternate logging session.
Create work tables	Work tables are created on each AMP for each target table. They will hold the DML steps to be performed as well as the input data to be applied.
Create error tables	Two error tables are created for each target table. One is for general errors and one is for uniqueness violations.
Create Restart log	Create the log for this run which will allow restarts.
Apply locks to tables	Utility locks are applied to the target table headers. This lock disallows any DDL to the table (except DROP).



Phase 2: DML Transaction

The second of IMPORT's five phases is the DML/Transaction. It performs the following:

Send prototype DML to the DBC

All DML statements (minus data) are sent from host to DBC where they are parsed. Steps are generated and stored on each AMP in the work table for the affected target table.

In Phase 1, work tables were created on each AMP for each target table. In this phase, the DML steps to be performed will be placed into the work tables.

Add a USING modifier to the request

Each request is submitted with a USING clause, with host data to be filled in at execution time.

Add a “Match Tag” to the request

Because it will be necessary to know which DML is to be associated with which incoming record (this is what the APPLY clause decides), we will use a “match tag” to link DML requests with input records.



Phase 2: DML Transaction

IMPORT
Phases

Send prototype DML to the Teradata Database

→ Store DML steps in work tables

Add a USING modifier to the request

→ Host data to be filled in from input file

Add a “Match Tag” to the request

→ Allows link between DML and transaction record

Phase 3: Acquisition

The third of **IMPORT**'s five phases is the Acquisition phase. It performs the following steps:

Get host data to the appropriate AMP worktables

Work tables only, not target tables, are involved in the Acquisition phase. The host reads the input file and tests for the APPLY conditions. A copy of the input record is made for every successful APPLY. The appropriate "match tag" information is also built into the record and the records are bunched into blocks. They are sent, round robin to the AMPs using a "quickpath," that is, they go through but are never processed by the PEs. The AMPs will have started "deblocker" tasks that will read the individual records from the block, hash on primary index value, and send that row to the AMP that holds the target row. The AMPs will also have started "receiver" tasks that pick up the incoming records with the correct hash value. These records are accumulated in the work table of the appropriate target table and reblocked. Records are built for the FALBACK subtables as well.

Sort the reblocked records in the work tables

Access locks are placed on the target tables. Records are sorted according to the hash value and the sequence in which they will be applied to the target table.

Set up transition to the Application phase

The Access lock on the target tables is upgraded to a Write lock. Utility locks are applied to the table headers indicating the Application phase is about to begin. An End Transaction statement commits all header changes for all target tables across all AMPs.



Phase 3: Acquisition

IMPORT Phases

- **Get the data from host and apply it to appropriate AMP worktables.**
 - Duplicate “input records” record for each successful APPLY.
 - Add “Match Tag” information to record.
 - Make blocks and send “quickpath” to AMPS.
 - Deblock and resend record to “correct” AMP.
- **Reblock and store in worktable of target table.**
 - Sort the reblocked records in the work tables.
 - Sort by hash value and sequence to be applied.
- **Set up transition to the Application phase.**
 - Upgrade locks on target tables to Write.
 - Set table headers for Application phase.
 - **This is effectively the “point of no return”.**

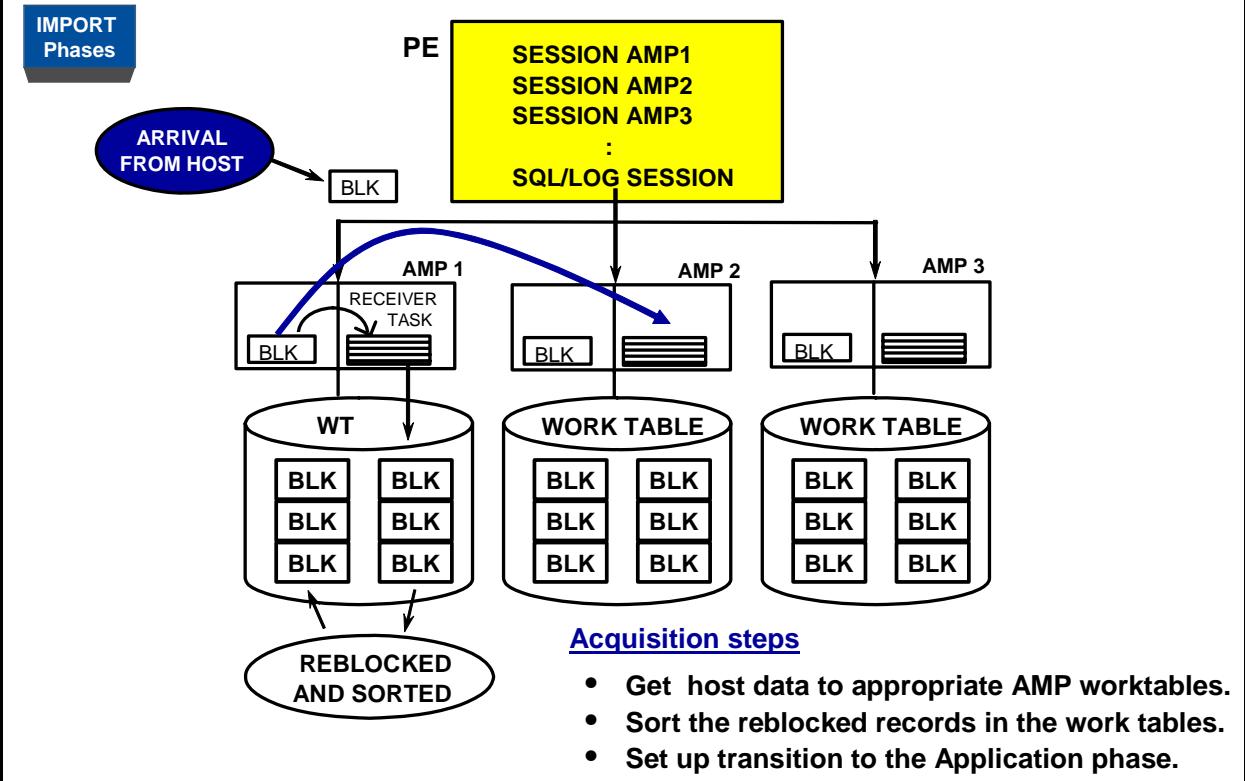
Notes:

- Errors that occur in this phase go into the Acquisition Error Table (default name is **ET_tablename**).
- There is no acquisition phase activity for a DELETE Task.

Phase 3: Acquisition – a Closer Look

The diagram on the facing page shows data movement from the host to the deblockers on to the appropriate receivers (based on hash code) then to the work table where finally it is sorted and reblocked.

Phase 3: Acquisition – a Closer Look



Phase 4: Application

The fourth of **IMPORT**'s five phases is the **Application phase**. It performs the following steps:

Execute MLOAD for each target table as a single multi-statement request

There is no further interaction with the host until the end of the phase. There is a separate execution of MLOAD for each target table, which means that the AMPs may independently and asynchronously apply changes to target tables. Because all EXEC MLOADs (up to five) are submitted as a multi-statement request, they are looked upon as a single transaction. If the transaction fails, changes are not rolled back, and the transaction is restartable at the point of failure. This eliminates the need for transient journaling.

Apply work subtable changes to target subtables

Each target table block requiring change is read and written only once. After reading the target block, that part of the work table (called a “work unit”) having matching hash codes is also read. Changes are applied to the target rows of the block according to the **DML** operation identified by that row's match tag.

If an error results from applying a row, that row is inserted into the UV error table associated with the target table for which that row was intended. Duplicate rows, missing update, or delete rows may also be inserted into this error table according to options specified by the user.

Because of the possibility of UPSERT processing and/or missing rows, it may be necessary to sweep the block more than once. A bit map is maintained showing which changes in the work unit have been applied and which have not. Once all processing has been done, the block is written out and a checkpoint is written to the work table.

After applying all changes to the target tables, **NUSI** changes, both primary and fallback, are applied to the target **NUSI** subtables. If the target table has permanent journaling, a Private Permanent Journal is maintained by MLOAD, and is then transferred to the true Permanent Journal.



Phase 4: Application

**IMPORT
Phases**

- Execute MLOAD for each target table as a single multi-statement request.
 - End of host interaction until end of phase.
 - AMPs independently apply changes to target tables.
 - Executed as a **single transaction** without rollback.
 - Restartable based on last checkpoint.
 - No transient journal needed.

Note:

- Errors that occur in this phase go into the Application Error Table (default name is UV_tablename).

Phase 4: Application – a Closer Look

The diagram on the facing page is intended to show how an individual AMP applies changes from the work subtables to the target tables during the **Application phase**.

There will be one MLOAD task in each AMP for each target table. If a single execution of MultiLoad is running against three target tables, there will be three tasks (out of a possible 60) being used for MultiLoad operations. Multiple **MultiLoads** on the system at one time will multiply this factor accordingly.

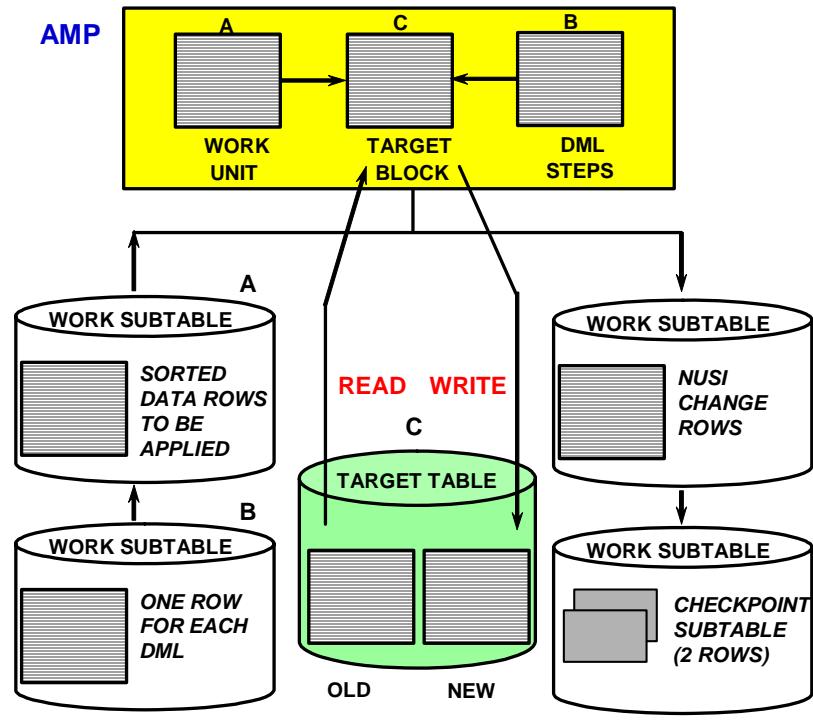
Two **MultiLoads** running, each against three target tables will result in each AMP having six MLOAD tasks running.

Phase 4: Application – a Closer Look

IMPORT Phases

Apply work subtable changes to target subtables:

- Affected blocks read/written only once.
- Changes applied based on matching row-hash.
- Errors written to appropriate error table.
- Checkpoint after writing each block.
- NUSI subtable changes applied.



Phase 5: Cleanup

The fifth **IMPORT** phase is the **End or Cleanup phase**, which performs the following steps:

Execute END MLOAD processing as an explicit transaction

After all changes have been applied to the target tables, many housekeeping chores remain before the utility is finished. All locks, both utility and DBC locks, must be released. All table headers must be restored to their original status across all AMPs. All Work Tables and any empty Error Tables are dropped. The dictionary cache for Target Tables is spoiled. Statistics are reported and a final error code is returned to the user. If the error code is zero, the Log Table is dropped.

MLOAD Session Logoff

A LOGOFF request is sent to each Load Control Task on an AMP that owns a session.



Phase 5: Cleanup

**IMPORT
Phases**

- Execute END MLOAD processing as a series of transactions performed by the host utility:
 - All locks are released.
 - Table headers are restored across all AMPs.
 - Dictionary cache of Target Tables is spoiled.
 - Statistics are reported.
 - Final Error Code is reported.
 - Target tables are made available to other users.
 - Work Tables are dropped.
 - Empty Error Tables are dropped.
 - Log Table is dropped (if Error Code = 0).
- MLOAD Session Logoff:
 - LOGOFF request is sent to each AMP with a session.

Sample MultiLoad DELETE Tasks

The following is an explanation of the components of a **MultiLoad DELETE** Task script:

.LOGTABLE defines the name of the Restart Log.

.LOGON defines username that will own the sessions.

.BEGIN DELETE MLOAD TABLES defines table that will participate in the MultiLoad.

.LAYOUT defines the layout of the incoming record.

.FIELD defines the name of an input field, its position in the record, and its data type.
(Absolute positioning may be done with a number, or relative positioning with an asterisk “*.”)

DELETE FROM standard SQL **DELETE** statement.

.IMPORT INFILE references **DDNAME** of the input file.

LAYOUT references previously defined **LAYOUT**.

.END MLOAD; defines end of MultiLoad script.

.LOGOFF; terminate the sessions.

A **DELETE** task is simpler than most **IMPORT** tasks. Note also that a **DELETE** task has no **.DML** and no **APPLY** clauses, because the single imported data record is unconditionally applied by the single **DELETE** statement.



Sample MultiLoad DELETE Tasks

Hard code the values of rows to be deleted.



```
.LOGTABLE Logtable002_mld;
.LOGON tdp3/user2,tyler;
.BEGIN DELETE MLOAD TABLES Employee;
DELETE FROM Employee WHERE Term_date > 0;
.END MLOAD;
.LOGOFF;
```

Pass a single row containing value(s) to be used.



```
.LOGTABLE Logtable003_mld;
.LOGON tdp3/user2,tyler;
.BEGIN DELETE MLOAD TABLES Employee;
.LAYOUT Remove;
.FIELD in_Termdate * INTEGER;
DELETE FROM Employee WHERE Term_date > :in_Termdate;
.IMPORT INFILE infile2
  LAYOUT Remove;
.END MLOAD;
.LOGOFF;
```

DELETE Task Differences from IMPORT Task

DELETE tasks operate very similarly to **IMPORT** tasks with some differences.

Differences include:

- Deleting based on a Unique Primary Index access is not permitted.
- A single **DML DELETE** statement is sent to each AMP with a match tag parcel.
- There is no **Acquisition phase** because there are no varying input records to apply.
- The **Application phase** reads each target block and deletes qualifying rows.
- Altered blocks are written back to disk.
- All other aspects of task are similar to **IMPORT** task.



DELETE Task Differences from IMPORT Task

DELETE tasks operate very similarly to IMPORT tasks with some differences:

- Deleting based on an equality UPI value is not permitted.
 - An inequality (e.g., $>$) test of a UPI value is permitted.
 - An equality (e.g., $=$) test of a NUPI value is permitted.
- A DML DELETE statement is sent to each AMP with a match tag parcel.
- No Acquisition phase because no variable input records to apply.
- Application phase reads each target block and deletes qualifying rows.
- All other aspects similar to IMPORT task.

Why use MultiLoad DELETE (versus SQL DELETE)?

- MultiLoad DELETE is faster and uses less disk space and I/O (no Transient Journal).
- MultiLoad DELETE is restartable.
 - If SQL DELETE is aborted, Teradata applies Transient Journal rows. SQL DELETE can be resubmitted, but starts from beginning.

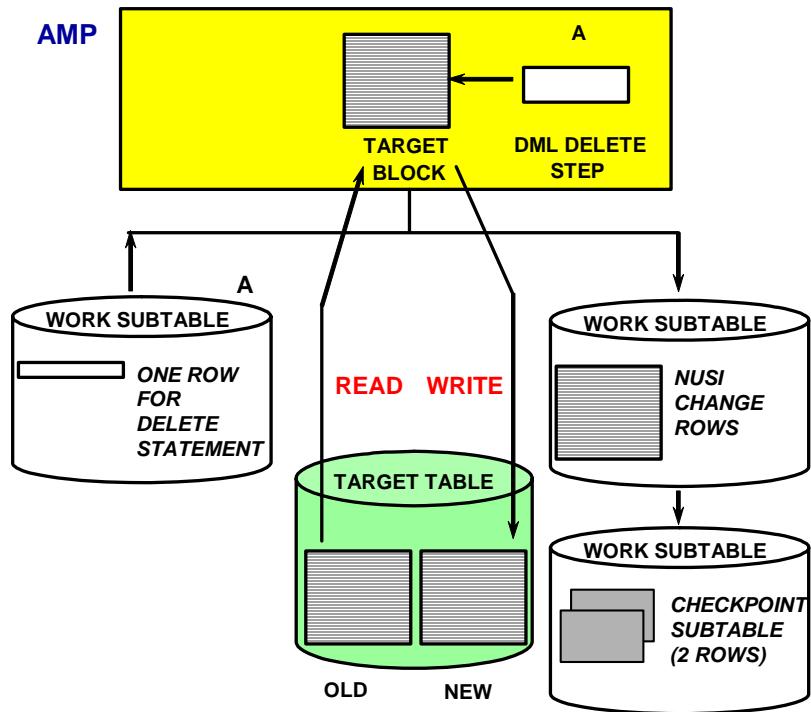
A Closer Look at **DELETE** Task Application Phase

The accompanying diagram attempts to show the movement of data in the Application phase of a **DELETE** task. Notice the absence of a Work Table carrying imported rows. There is no work table because the same **DML DELETE** statement will be applied to every row in the table.

Using MultiLoad **DELETE** tasks give you an advantage over using traditional utilities to accomplish a similar **DELETE** since there is no use of a transient journal, and no rollback in the event of failure. Because of the restart capabilities of MultiLoad, no completed work needs to be reapplied.

A Closer Look at DELETE Task Application Phase

- Note absence of Work Table for Import rows.
- Faster than traditional SQL DELETE due to:
 - Lack of transient journaling
 - No rollback of work
 - Restartable from checkpoint



MultiLoad Locks

MultiLoad uses several different locks at various stages of the operation. These are intended to insure maximum availability of the target tables during **MultiLoad** processing as well as restartability of various phases of the utility.

Utility locks

Utility locks are placed in the table headers to indicate to utilities such as Reconfig and Rebuild that an MLOAD is in progress and to do special processing. Utility locks are the minimum level of lock required when an MLOAD is invoked even when it is not currently running.

There are two types of utility locks: **Acquisition locks** and **Application locks**. They are defined below:

- Acquisition locks** prevent any **DDL** activity against the table with the exception of the **DROP** command, but does allow all **DML** command access.
- Application locks** allow concurrent access-lock **SELECT** access and the **DROP DDL** statement, but reject all other **DML** and **DDL** statements.

There are two important points to bear in mind in understanding the locking strategy of MultiLoad. First, there are never any **Exclusive locks** used on the target tables. This means that **Access locks** are always useable against target tables throughout the MultiLoad execution.

Secondly, each of the five stages of MultiLoad is treated as one or more DBC transactions. This means that the end of each stage is also the end of a transaction and that all locks associated with that stage are thereby released. New locks are applied with the beginning of the next phase/transaction. This permits the ability of other transactions, outside of MultiLoad, to effect the target table rows during a MultiLoad execution.

For example, at the end of the **Acquisition phase**, the access lock on the target table is released. As the **Application phase** begins, a “new write lock” is applied to the target table as a part of the new transaction. Between these two locks, other external requests may “get in” to the target table, thus preventing them from having to wait in the queue until MultiLoad completes.

Note: If you need to examine the logtable, the workfiles, or the error files at any time during the execution of MultiLoad, you MUST use an ACCESS lock to access them in order to prevent MultiLoad from abnormally terminating due to locking problems.



MultiLoad Locks

Utility locks: Placed in table headers to alert other utilities that a MultiLoad is in session for this table. They include:

- **Acquisition lock**
DML — allows all
DDL — allows DROP only
- **Application lock**
DML — allows SELECT with ACCESS only
DDL — allows DROP only

Restarting MultiLoad

MultiLoad contains a number of features that allow for recovery from any host or DBC failure. It does this with minimal requirements for job resubmission or continuation. Upon restart, MultiLoad will check the restart log table and resume operations from where it had previously left off.

If a **DBC restart** occurs during MLOAD, the host program will reinitiate MLOAD after DBC recovery and continue from where it left off with no user interaction required.

If a **host restart** occurs during MLOAD, or the job is aborted, the user may resubmit the script as-is, and MLOAD will determine its stopping point and begin again. No script alteration is required.

If an MLOAD task is stopped during the Application phase, it must be resubmitted and allowed to run to completion.

Restarts are initiated based on checkpoint information in the Logtable. Because MLOAD does not do transient journaling, a traditional rollback operation is not performed when a failure occurs. MLOAD is designed with a checkpointing feature that allows for restart of the job with minimal loss of work. The following principles guide the MultiLoad checkpointing strategy:

Acquisition phase checkpointing is performed according to user specification as specified in the **.BEGIN MLOAD** statement. This checkpoint can be based on time or on number of records processed. The default checkpoint interval is fifteen minutes.

Application phase checkpointing is performed each time a data block is written to the target table. Each block is written one time.

Sort phase(s) sort operations do their own internal checkpointing that overrides the higher level checkpoints.



Restarting MultiLoad

Teradata Restart

- MLOAD reinitiated automatically after Teradata recovery.
- Continue from checkpoint without user interaction.

Host restart

- Resubmit the original script.
- MLOAD determines its stopping point and restarts.

MultiLoad Checkpointing Strategy

Acquisition phase

- Checkpointing is performed according to user.
- Checkpoint based on time or on number of records.
- Default checkpoint interval is fifteen minutes.

Application phase

- Checkpointing done after each write of data block.
- Each block is written at most only one time.

Sort phase(s)

- Sort operations do their own internal checkpointing.

RELEASE MLOAD Statement

Once MultiLoad execution has begun, table headers are updated in the target tables indicating that a MLOAD is in progress. Even if the MLOAD doesn't successfully complete, target tables are still considered under the control of the MLOAD and access to them will be restricted accordingly.

The **RELEASE MLOAD** statement provides a way to return tables to general availability where there is no desire to restart the MLOAD. If the specified table is in the Preliminary, DDL or the early part of the Acquisition phase, the **RELEASE MLOAD** statement makes the table completely accessible and prevents any attempt to restart the MLOAD. If the MLOAD had proceeded into the Application phase, the **RELEASE MLOAD** statement is rejected and the job must be restarted.

Once a lock has been applied to the target table, the **RELEASE MLOAD** statement will not be effective until the transaction with the lock completes. Even if the transaction completes, **RELEASE MLOAD** may be rejected if the point of no return has occurred. In a **DELETE** task, because there is no Acquisition phase, the point of no return is the point in the DML phase when the **DELETE** statement is sent to the DBC. In an **IMPORT** task, the actual point of no return is the point at which the Acquisition phase ends.

To successfully complete a **RELEASE MLOAD**, the following procedure must be followed:

1. Make sure MLOAD is not running; abort it if it is. (Note: MLOAD is still in a restartable state if aborted. If it is past the point of no return, go to step 4.)
2. Enter **RELEASE MLOAD**.
3. If successful, drop the work and error tables. (You may wish to examine any errors in the error tables before dropping them.)
4. If not successful, determine if past point of no return. If so, either restart MLOAD and let it complete, or drop target, work, and error tables. Otherwise, handle errors as appropriate.



RELEASE MLOAD Statement

RELEASE MLOAD Employee, Job, Department;

- Returns target tables to general availability.
- Prevents any attempt to restart MultiLoad.
- Cannot be successful in all cases.
- Cannot override a target table lock.
- IMPORT — possible before Application phase.
- DELETE — possible during Preliminary phase.

To successfully complete a RELEASE MLOAD:

1. Make sure MLOAD is not running; abort if it is. (If it is past the point of no return, go to step 4.)
2. Enter RELEASE MLOAD.
3. If successful, drop the log, work, and error tables.
4. If not successful:
 - a.) restart MLOAD and let it complete, or
 - b.) drop target, work, and error tables, or
 - c.) handle error as appropriate.

Invoking MultiLoad

The facing page displays the commands used to invoke the MultiLoad utility in batch mode. The parameters for each command are listed in the three-column table.

Optionally, when MultiLoad starts, it can read a configuration file to establish defaults for the MultiLoad job. On network-attached systems (e.g., UNIX and Windows), MultiLoad can read configuration parameters from a file named **mloadcfg.dat**. This file is located either in the current directory or the director referenced by the variable MLOADLIB.

On channel-attached systems, the DD statement for the MultiLoad configuration file must be labeled MLOADCFG.

There are 7 parameters you can set in this file.

- CHARSET=character-set-name
- ERRLOG=filename
- BRIEF=on/off
- MAXSESS=max-sessions
- MINSESS=min-sessions
- MATCHLEN=ON/OFF
- DATAENCRYPTION=ON/OFF

Note: When you enable MATCHLEN, MultiLoad verifies that the record length of the import data is the same as the layout record length specified by the IMPORT command.

The values that you specify in the MultiLoad configuration file override the internal utility default values for these parameters. Configuration file parameters can be overridden with runtime parameters. The order of preference (highest to lowest) for these parameters is:

- 1 – Runtime parameters
- 2 – MultiLoad script parameters
- 3 – Configuration file parameters
- 4 – MultiLoad default values

The MultiLoad utility automatically checks for a configuration file each time you invoke the utility. Upon locating a configuration file, the utility sets the defaults as specified, produces the appropriate output messages and begins processing your MultiLoad job.

If the configuration file cannot be opened, or if the MultiLoad utility encounters syntax errors in the file, the utility produces an output message, disregards the error condition and begins processing your MultiLoad job. An invalid configuration file entry does not abort your MultiLoad job.

If there is no configuration file, the utility begins processing your MultiLoad job without an error indication. The configuration file is an optional feature of the MultiLoad utility, and its absence is not considered to be an error condition.



Invoking MultiLoad

Network Attached Systems: mload [PARAMETERS] < *scriptname* >*outfilename*

Channel-Attached MVS Systems: // EXEC TDSMLOAD MLPARM= [PARAMETERS]

Channel-Attached VM Systems: EXEC MLOAD [PARAMETERS]

Channel Parameter	Network Parameter	Description
BRIEF	-b	Reduces print output runtime to the least information required to determine success or failure.
CHARSET= <i>charsetname</i>	-c <i>charsetname</i>	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets.
ERRLOG= <i>filename</i>	-e <i>filename</i>	Alternate file specification for error messages; produces a duplicate record.
" <i>multiload command</i> "	-r ' <i>multiload cmd</i> '	Signifies the start of a MultiLoad job; usually a RUN FILE command that specifies the script file.
MAXSESS= <i>max sessions</i>	-M <i>max sessions</i>	Maximum number of MultiLoad sessions logged on.
MINSESS= <i>min sessions</i>	-N <i>min sessions</i>	Minimum number of MultiLoad sessions logged on.
	< <i>scriptname</i>	Name of file that contains MultiLoad commands and SQL statements.
	> <i>outfilename</i>	Name of output file for MultiLoad messages.

Application Utility Checklist

The facing page adds the MultiLoad capabilities to the checklist.



Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED	Yes (SE)	Yes (SE)	
DML Functions	ALL	INSERT	SELECT	INS/UPD/DEL	
Multiple DML	Yes	No	Yes	Yes	
Multiple Tables	Yes	No	Yes	Yes	
Multiple Sessions	Yes	Yes	Yes	Yes	
Protocol Used	SQL	FASTLOAD	EXPORT	MULTILOAD	
Conditional Expressions	Yes	No	Yes	Yes	
Arithmetic Calculations	Yes	No	Yes	Yes	
Data Conversion	Yes	1 per column	Yes	Yes	
Error Tables	No	Yes	No	Yes	
Error Limits	No	Yes	No	Yes	
User-written Routines	No	Yes	Yes	Yes	
Support Environment (SE)	No	No	Yes	Yes	

Summary

The facing page summarizes some of the important concepts regarding this module.



Summary

- Batch mode utility.
- Supports up to five populated tables.
- Multiple operations with one pass of input files.
- Conditional logic for applying changes.
- Supports INSERTs, UPDATEs, DELETEs and UPSERTs; typically with batch inputs from a host file.
- Affected data blocks only written once.
- Full Restart capability.
- Error reporting via error files.
- Support for INMODs.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Answer True or False.

1. True or False. With MultiLoad, you can import data from the host into populated tables.
2. True or False. MultiLoad cannot process tables with USIs or Referential Integrity defined.
3. True or False. MultiLoad allows changes to the value of a table's primary index.
4. True or False. MultiLoad allows you to change the value of a column based on its current value.
5. True or False. MultiLoad permits non-exclusive access to target tables from other users except during Application Phase.

Match the MultiLoad Phase in the first column to its corresponding task in the second column.

- | | |
|---|---|
| 1. <input type="text"/> Preliminary | A. Acquires or creates Restart Log Table. |
| 2. <input type="text"/> DML Transaction | B. Locks are released. |
| 3. <input type="text"/> Acquisition | C. Applies (loads) data to the work tables. |
| 4. <input type="text"/> Application | D. Execute mload for each target table as a single multi-statement request. |
| 5. <input type="text"/> Cleanup | E. Stores DML steps in work tables |

Lab Exercise 37-1

To create the **data37_1** file, execute the following statements in BTEQ.

```
.EXPORT DATA FILE=data37_1;  
EXEC AP.Lab37_1;  
.EXPORT RESET;
```

Note: Use the SHOW MACRO command to view precisely what is being exported to your transaction file (data37_1).

The size of data37_1 should be 2400 bytes long after executing the macro AP.Lab37_1.

The format of data37_1 is:

A	Integer
C	Integer
T	Integer

The control letter (A, C, or T) must be in upper-case letters in the APPL Y statements. The Integer value represents the Primary Index Value that you will use to delete a row in the appropriate table.

For example, if the input record contains a code of A, delete a row from the Accounts table using the Integer value as the PI value.

If the input record contains a code of C, delete a row from the Customer table using the Integer value as the PI value.

If the input record contains a code of T, delete a row from the Trans table using the Integer value as the PI value.

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab37_13.mld** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function
To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 37-1

Purpose

In this lab, you will use MultiLoad to delete rows from your three tables. An input file will be created which will contain a control letter (A - Accounts, C - Customer, and T - Trans) followed by a primary index value for the appropriate table.

What you need

Your three tables with two hundred (200) rows in each.

Tasks

1. Prepare the data file by executing the macro AP.Lab37_1. Export your data to a file called *data37_1*.
2. Prepare your tables by doing the following:
 - a. In BTEQ, issue a Delete All command on each of your tables.
 - b. While still in BTEQ, execute the following script which will load the specified 200 rows into each of the tables:

```
INSERT INTO Accounts  SELECT * FROM AP.Accounts      WHERE Account_Number LT 20024201;
INSERT INTO Customer  SELECT * FROM AP.Customer    WHERE Customer_Number LT 2201;
INSERT INTO Trans     SELECT * FROM AP.Trans       WHERE Account_Number LT 20024201;
```
3. Prepare your MultiLoad script to Delete Rows from each of the tables depending on the incoming code (A, C, or T) from *data37_1*. This job should result in deleting 100 rows from each of the three tables.
4. Check your results by doing a **SELECT COUNT(*)** on each of your tables.

Teradata Training

Notes

Module 38



A MultiLoad Application

After completing this module, you will be able to:

- **Describe the tables involved in a MultiLoad job.**
- **Set error limits as a record value or as a percentage of loaded rows.**
- **Specify a checkpoint interval.**
- **Redefine input record layout.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

New Accounts Application – Description	38-4
New Accounts Application Script (1 of 3).....	38-6
New Accounts Application Script (2 of 3).....	38-8
New Accounts Application Script (3 of 3).....	38-10
.BEGIN IMPORT Task Commands	38-12
Work Tables	38-14
Error Tables.....	38-16
ERRLIMIT	38-18
CHECKPOINT	38-20
More .BEGIN Parameters	38-22
SESSIONS <i>max min</i>	38-22
More .BEGIN Parameters: AMPCHECK.....	38-24
DELETE Task Commands.....	38-26
.LAYOUT and .TABLE.....	38-28
.LAYOUT Parameters — CONTINUEIF.....	38-30
.LAYOUT Parameters — INDICATORS	38-32
.FIELD and .FILLER	38-34
Performance Considerations	38-34
.LAYOUT Command — Examples	38-36
Redefining the Input – Example	38-38
The .DML Command Options	38-40
The .DML Command Options (cont.).....	38-42
MultiLoad Statistics	38-44
INMOD	38-46
Other IMPORT options.....	38-46
AXSMOD	38-46
Summary	38-48
Review Questions	38-50
Lab Exercise 38-1	38-52
Lab Exercise 38-2	38-54

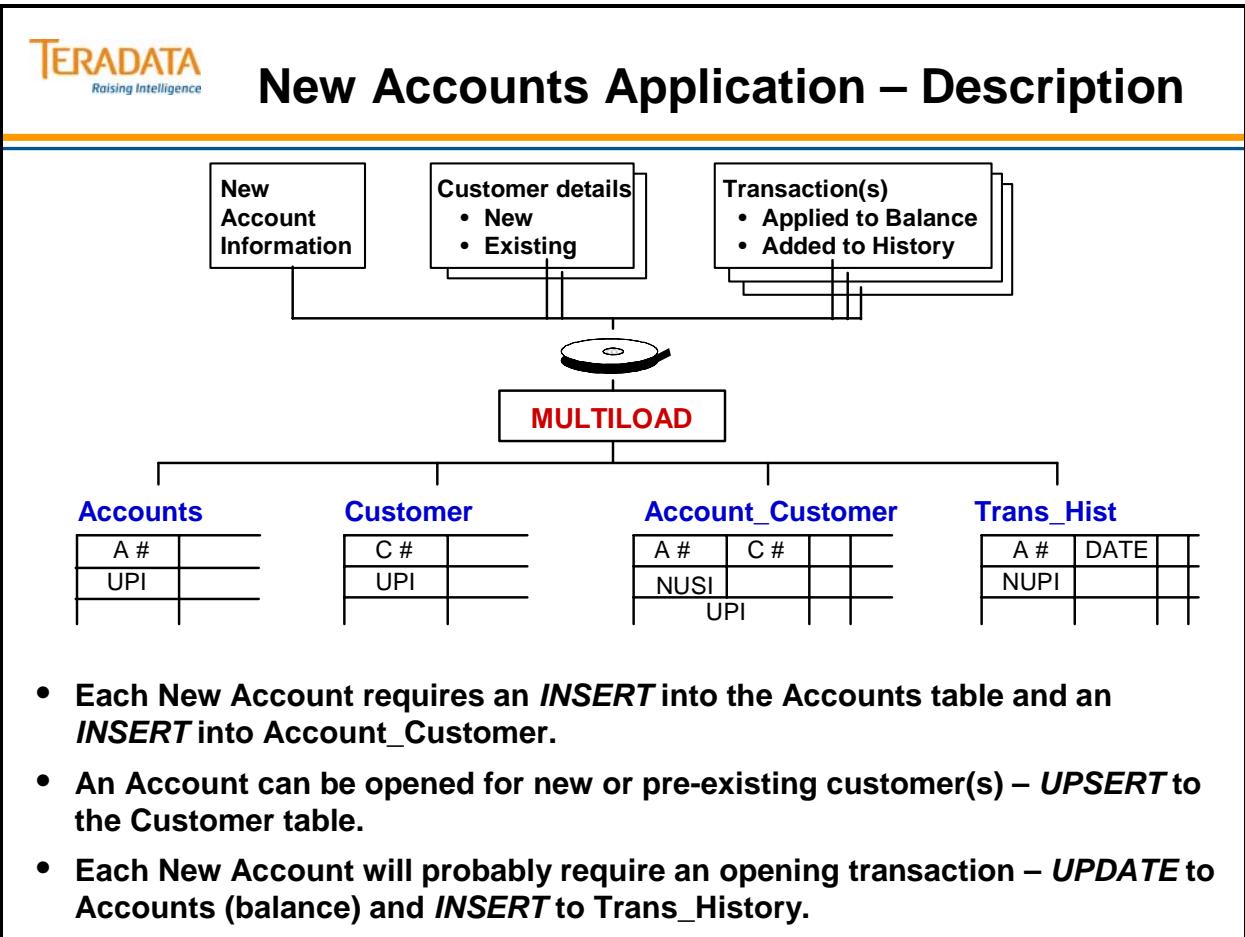
New Accounts Application – Description

The facing page diagrams an example of a bank procedure for customers opening new accounts. The application must be able to handle:

- New customers with new accounts
- Existing customers who open new accounts.

You need to do an UPSERT (an UPDATE or an INSERT). The tasks are to:

- Execute an UPDATE to a Customer Row. (This UPDATE actually only checks for the existence of the Customer Row.)
- If this fails, an INSERT of a Customer Row.



New Accounts Application Script (1 of 3)

The entire input script for this application is shown on the next few pages. The first page shows the Support Environment statements to define the LOGTABLE and LOGON.

The .BEGIN IMPORT statement defines the tables used in this example.

The .LAYOUT statement is followed by the definition of .FIELD or .FILLER statements.

These statements are discussed in more detail later in this module.



New Accounts Application Script (1 of 3)

```
.LOGTABLE Newacct_logtable_mld;
.LOGON tdpid/username, password;
.BEGIN IMPORT MLOAD TABLES Accounts, Account_Customer, Customer, Trans_Hist ;
.LAYOUT New_Acc_Layout ;
.FILLER in_Field_Indicator      1      CHAR(1) ;
.FIELD   in_Account_Number     2      INTEGER ;
.FIELD   in_Number             *      INTEGER ;
.FIELD   in_Street              *      CHAR(25) ;
.FIELD   in_City                *      CHAR(20) ;
.FIELD   in_State               *      CHAR(2) ;
.FIELD   in_Zip_Code            *      INTEGER ;
.FIELD   in_Balance_Forward    *      INTEGER ;
.FIELD   in_Balance_Current    *      INTEGER ;
.FIELD   in_Customer_Number    2      INTEGER ;
.FIELD   in_Last_Name           *      CHAR(25) ;
.FIELD   in_First_Name          *      CHAR(20) ;
.FIELD   in_Social_Security     *      INTEGER ;
.FIELD   in_AC_Account_Number  2      INTEGER ;
.FIELD   in_AC_Customer_Number *      INTEGER ;
.FIELD   in_Trans_Number        2      INTEGER ;
.FIELD   in_Trans_Account_Number *      INTEGER ;
.FIELD   in_Trans_ID            *      INTEGER ;
.FIELD   in_Trans_Amount         *      INTEGER ;
```

New Accounts Application Script (2 of 3)

The facing page shows all the .DML statements that define labels for the one or more DML commands that follow.

The sequence in which these commands are performed is indicated by the APPLY clauses in the .IMPORT statement. APPLY clauses may contain conditions to be met before the command is applied to the input data. The .IMPORT statement also defines the name of the file that contains the input data and the LAYOUT for that file. The LAYOUT is matched to a defined .LAYOUT statement by the logical name that was associated with it.

The MultiLoad script is terminated with the .END MLOAD statement.



New Accounts Application Script (2 of 3)

```

.DML LABEL Label_A ;
  INSERT INTO Accounts VALUES
    ( :in_Account_Number, :in_Number, :in_Street, :in_City ,:in_State, :in_Zip_Code,
     :in_Balance_Forward, :in_Balance_Current ) ;

.DML LABEL Label_B ;
  INSERT INTO Account_Customer VALUES
    ( :in_AC_Account_Number, :in_AC_Customer_Number);

.DML LABEL Label_C MARK MISSING UPDATE ROWS DO INSERT FOR MISSING UPDATE ROWS ;
  UPDATE Customer SET Last_Name = :in_Last_Name
    WHERE Customer_Number = :in_Customer_Number ;
  INSERT INTO Customer VALUES
    ( :in_Customer_Number, :in_Last_Name, :in_First_Name, :in_Social_Security ) ;

.DML LABEL Label_D ;
  INSERT INTO Trans_Hist VALUES
    ( :in_Trans_Number, DATE, :in_Trans_Account_Number, :in_Trans_ID, :in_Trans_Amount ) ;
  UPDATE Accounts
    SET Balance_Current =Balance_Current + :in_Trans_Amount
    WHERE Account_Number =:in_Trans_Account_Number ;

.IMPORT INFILE datain4 LAYOUT New_Acc_Layout
  APPLY Label_A WHERE in_Field_Indicator = 'A'
  APPLY Label_B WHERE in_Field_Indicator = 'B'
  APPLY Label_C WHERE in_Field_Indicator = 'C'
  APPLY Label_D WHERE in_Field_Indicator = 'D' ;

.END MLOAD ;

```

New Accounts Application Script (3 of 3)

The facing page shows the support environment commands to check MultiLoad support environment counts and COLLECT STATISTICS on the tables as part of the utility job.

System variables that are available with MultiLoad include:

&SYSDELCNT(n)	Delete Count	Ex., &SYSDELCNT1, ...
&SYSINSCNT(n)	Insert Count	Ex., &SYSINSCNT1, ...
&SYSUPDCNT(n)	Update Count	
&SYSETCNT(n)	Error Table Count	
&SYSUVCNT(n)	Uniqueness Violation Count	
&SYSRCDCNT(n)	Count of import records read	
&SYSRJCTCNT(n)	Count of records rejected from import file	

Note: n is 1 to 5

A .LOGOFF; statement terminates the Support Environment session.



New Accounts Application Script (3 of 3)

```

.IF (&SYSINSCNT1 > 0 OR &SUPDCNT1 > 0) THEN;
  COLLECT STATISTICS ON Accounts;
.ENDIF;

.IF (&SYSINSCNT2 > 0) THEN;
  COLLECT STATISTICS ON Account_Customer;
.ENDIF;

.IF (&SYSINSCNT3 > 0 OR &SUPDCNT3 > 0) THEN;
  COLLECT STATISTICS ON Customer;
.ENDIF;

.IF (&SYSINSCNT4 > 0) THEN;
  COLLECT STATISTICS ON Trans_Hist;
.ENDIF;

.LOGOFF ;

```

MultiLoad environment variables can be checked to optionally COLLECT STATISTICS as part of the job.

&SYSDELCNT_	where _ is 1 to 5
&SYSINSCNT_	"
&SUPDCNT_	"
&SYSETCNT_	"
&SYSUVCNT_	"
&SYSRDCDCNT_	"
&SYSRJCTCNT_	"

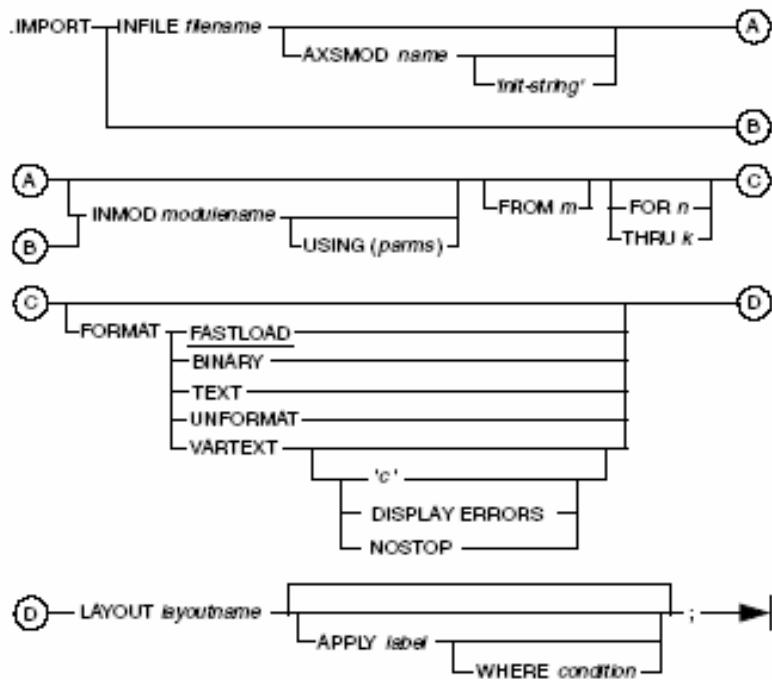
Note:

.BEGIN IMPORT MLOAD TABLES Accounts, Account_Customer, Customer, Trans_Hist ;

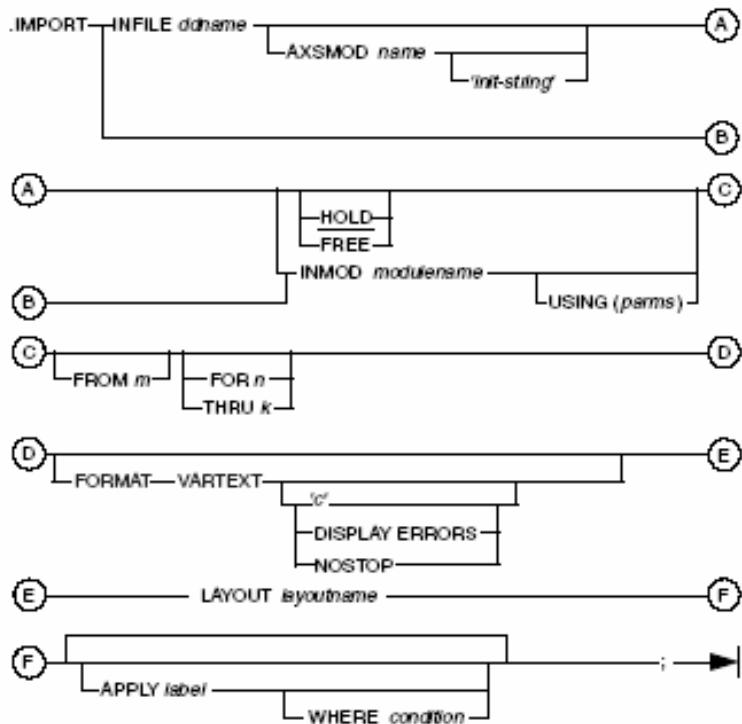
↑ ↑ ↑ ↑
 1st table 2nd table 3rd table 4th table

.BEGIN IMPORT Task Commands

The format of the .IMPORT command for a network-attached client system is:



The format of the .IMPORT command for a channel-attached client system is:





.BEGIN IMPORT Task Commands

- Specifies the tables and optionally the work and error tables used in this MultiLoad job.
- Also used to specify miscellaneous MultiLoad options such as checkpoint, sessions, etc.

```
.BEGIN [IMPORT] MLOAD
  TABLES      tname1, tname2, ...
  WORKTABLES  wt_table1, wt_table2, ...
  ERRORTABLES et_table1 uv_table1, et_table2 uv_table2, ...
  ERRLIMIT    errcount [errpercent]
  CHECKPOINT  rate                      (Default – 15 min.)
  SESSIONS    max [min]                  (Default – 1 per AMP + 2)
  TENACITY    hours                     (Default – 4 hours)
  SLEEP       minutes                   (Default – 6 minutes)
  AMPCHECK   NONE | APPLY | ALL
  NOTIFY     OFF | LOW | MEDIUM | HIGH ...
;
.END MLOAD ;
```

Work Tables

There must be one work table for each data table.

If WORKTABLES are not defined, these are created with the table name prefixed by 'WT_'.

If the data table is Fallback protected, then the associated Work table is Fallback protected.

If the data table is not Fallback protected, then the associated Work table is not Fallback protected.



Work Tables

IMPORT Task: **WORKTABLES *wt_table1*, *wt_table2*, ...**

.BEGIN
parameters

DELETE Task: **WORKTABLES *wt_table1***

- Default is in user's default database and the work table is named ***WT_TableName***.
- Alternative may be specified as ***DataBaseName.WorkTableName***.
- There must be one work table defined for each data table.

Example:

```
.BEGIN [IMPORT] MLOAD
  TABLES      Employee, PayCheck
  WORKTABLES  util_db.WT_Emp
               ,util_db.WT_Pay
  ERRORTABLES util_db.ET_Emp util_db.UV_Emp
               ,util_db.ET_Pay  util_db.UV_Pay
  ....;
```

The Error Tables are described on the next page.

Error Tables

ERRORTABLES default to an ‘ET_’ or ‘UV_’ prefix and the Table name. There will be two error tables for each user table.

The “ET_tablename” error table is referred to as the Acquisition Phase Error Table. Errors that occur in the Acquisition Phase are placed in this table.

The “UV_tablename” error table is referred to as the Application Phase Error Table. Errors that occur in the Application Phase are placed in this table.

Regardless if the data table is Fallback or No Fallback, the ET and UV error tables are automatically Fallback protected.



Error Tables

ERRORTABLES et_tab1 uv_tab1, et_tab2 uv_tab2, ...

.BEGIN
parameters

- Error table 1 (et)
 - default is the user's database and the table is named **ET_Tablename**.
 - contains any errors that occur in the **Acquisition Phase**.
 - contains primary index overflow errors that occur in the **Application phase**
- Error table 2 (uv)
 - default is the user's database and the table is named **UV_Tablename**.
 - contains **Application Phase** errors.
 - Uniqueness violations
 - Constraint errors
 - Overflow errors on columns other than primary index

Example:

```
.BEGIN [IMPORT] MLOAD
  TABLES      Employee, PayCheck
  WORKTABLES  util_db.WT_Emp
              ,util_db.WT_Pay
  ERRORTABLES util_db.ET_Emp util_db.UV_Emp
              ,util_db.ET_Pay util_db.UV_Pay
  ....;
```

ERRLIMIT

The ERRLIMIT option allows you to specify an error count and, optionally, an error percent.

The specification of an *error count* indicates the approximate number of errors (not uniqueness violations) that should cause the MultiLoad to stop processing (and abort).

The additional definition of an *error percent* indicates that you wish to stop processing when a percentage of errors has been detected after an approximate number of records have been transmitted.



ERRLIMIT

ERRLIMIT ErrCount

.BEGIN
parameters

Without ERRPERCENT:

- Specifies approximate number of data errors permitted during Acquisition.
- Does not count Uniqueness violations.

ERRLIMIT ErrCount ErrPercent

With ERRPERCENT:

- Specifies a percentage of data errors after an approximate number of records has been transmitted.

Example: **ERRLIMIT 10000 5**

In this example, after processing 10,000 input records, the system looks for an error rate of 5%.

CHECKPOINT

The CHECKPOINT option defines the checkpoint rate as either the *number of records* or a *time interval*.

If you specify a CHECKPOINT *rate* of 60 or more, a checkpoint operation occurs after each multiple of that number of records is processed.

If you specify a CHECKPOINT *rate* of less than 60, a checkpoint operation occurs at the specified frequency, in minutes.

The default is for MultiLoad to perform a CHECKPOINT every 15 minutes. If you do not use the CHECKPOINT *rate* specification, the MultiLoad utility performs a checkpoint operation at the default rate — every 15 minutes and at the end of each phase.

Note: Specifying a CHECKPOINT rate of 0 inhibits the checkpoint function—the MultiLoad utility does not perform any checkpoint operations during the import task.



CHECKPOINT

- Rate may be specified in the Acquisition Phase of a complex IMPORT task as:
 - A *number of incoming records* (exact count; not less than 60)
 - A *time interval in minutes* (approximate; less than 60)
- If no CHECKPOINT value is specified MultiLoad will checkpoint every 15 minutes, and at the end of each Phase. The default is 15 minutes.

.BEGIN
parameters

Example 1: **CHECKPOINT 30**

In this example, a 30-minute time interval is specified.

Example 2: **CHECKPOINT 100000**

In this example, a checkpoint after 100,000 input records is specified.

More .BEGIN Parameters

The facing page specifies additional parameters you can use with the .BEGIN statement.

SESSIONS *max min*

max maximum number of MultiLoad sessions that will be logged on when you enter a LOGON.

- The *max* specification must be greater than zero.
- If you specify a SESSIONS *max* value that is larger than the number of available AMPs, the MultiLoad utility limits the sessions to one per working AMP.
- The default maximum is one session for each AMP.
- Using the asterisk character as the *max* specification logs on for the maximum number of sessions—one for each AMP.

min optional, the minimum number of sessions required to run the job.

- The *min* specification must be greater than zero.
- The default minimum, if you do not use the SESSIONS option or specify a min value, is 1.
- Using the asterisk character as the *min* specification logs on for at least one session, but less than or equal to the max specification.



More .BEGIN Parameters

SESSIONS

SESSIONS 48 32

.BEGIN
parameters

- Used to specify the maximum, and optionally, minimum sessions generated by MultiLoad.

TENACITY

TENACITY 10

- Number of hours MultiLoad will try to establish a connection to the system.
- The default is 4 hours.

SLEEP

SLEEP 3

- Number of minutes MultiLoad waits before retrying a logon; must be greater than 0.
- The default is 6 minutes.

NOTIFY

NOTIFY OFF

Note: The MultiLoad manual specifies in detail which events are associated with each level.

- NOTIFY LOW** for initialize event and CLIV2 errors.
- NOTIFY MEDIUM** for the most significant events.
- NOTIFY HIGH** for every MultiLoad event that involves an operational decision point.
- NOTIFY OFF** suppresses the notify option.

More .BEGIN Parameters: AMPCHECK

Use the AMPCHECK option to specify what you want to occur in the event of an AMP being down.

The default (APPLY) specifies that if AMPs are offline, you want MultiLoad to continue processing every phase except the Application phase as long as all tables involved in the MultiLoad job have been defined with FALLBACK protection.



More .BEGIN Parameters: AMPCHECK

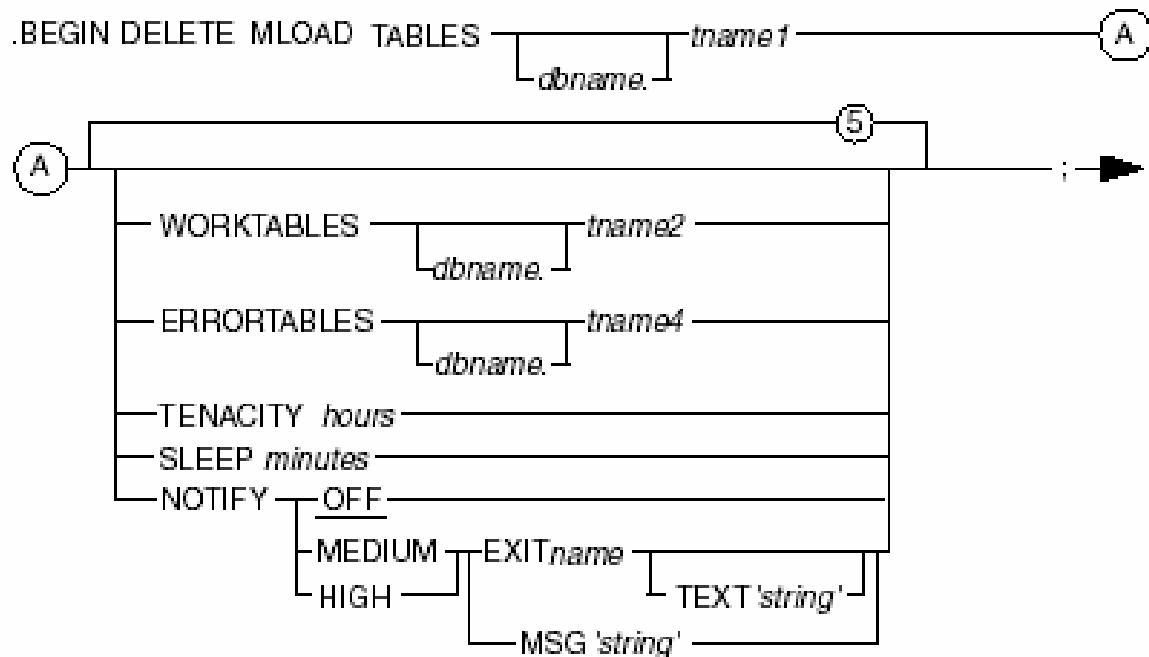
AMPCHECK NONE | APPLY | ALL

.BEGIN
parameters

- **NONE**
 - MultiLoad will not perform an AMPCHECK.
 - It *will proceed* if AMPs are offline, provided all target tables are Fallback.
- **APPLY**
 - MultiLoad will continue in all phases except the Application phase with AMPs offline, provided all target tables are Fallback.
 - **This is the default.**
- **ALL**
 - MultiLoad *will not proceed* with down AMPs, regardless of the protection-type of the target tables.
 - **Most conservative option.**

DELETE Task Commands

The facing page summarizes the options for the DELETE task. Note that it has many of the same options as the IMPORT task.





DELETE Task Commands

- Specifies the table and optionally the work and error tables used in this MultiLoad Delete task.
- Also used to specify miscellaneous MultiLoad options such as tenacity, sleep, etc.

```
.BEGIN DELETE MLOAD
    TABLES          tname1
    WORKTABLES      wt_table1
    ERRORTABLES    et_table1
    TENACITY        hours           (Default – 4 hours)
    SLEEP           minutes         (Default – 6 minutes)
    NOTIFY          OFF | LOW | MEDIUM | HIGH ...
;
.END MLOAD ;
```

.LAYOUT and .TABLE

The layout is given a name that is referenced in the .IMPORT statement.

The .LAYOUT statement can be used multiple times within the same MultiLoad job to indicate files with different fields. The layout name specifies the layout that should be used for the APPLY clauses which follow.

The .LAYOUT statement is always followed by either a .TABLE or .FIELD/.FILLER statement. If .TABLE is used, the input data type and fields are the same as an existing database table definition.



.LAYOUT and .TABLE

.LAYOUT

- Must appear between the .BEGIN IMPORT MLOAD command and the applicable .IMPORT command.
- Must be immediately followed by .TABLE, .FIELD, or .FILLER commands.

.TABLE

- The input fields are defined with the same name, data type, and order of an existing table.

Format:

```
.LAYOUT  layoutname
        [ CONTINUEIF position = variable ]
        INDICATORS

.TABLE   tableref ;
```

.LAYOUT Parameters — CONTINUEIF

Input data records may be concatenated if the record to be concatenated follows the initial data portion. For example, a data set or file contains records that are currently divided into two or three parts (records), but need to be treated as one record for MultiLoad. The CONTINUEIF option is used to indicate the value to check for when concatenating records.

The CONTINUEIF option must be of the type CHARACTER and may be multiple characters.

The general format is:

CONTINUEIF *position* = *value*

position

an unsigned integer (never an asterisk) that specifies the starting character position of the field of every input record that contains the continuation indicator. **Note:** The position is relative to the first character position of the input record or input record fragment, which is always position 1.

value

the continuation indicator specified as a character constant or a string constant. The MultiLoad utility uses the length of the constant as the length of the continuation indicator field.

Miscellaneous Notes:

- The ***value*** is case sensitive – always specify the correct character case for this parameter.
- If the conditional phrase is “true,” then the MultiLoad utility forms a single record to be sent to the Teradata RDBMS by concatenating the next input record at the end of the current record. (The current record is the one most recently obtained from the external data source.)
- If the conditional phrase is “false”, then the MultiLoad utility sends the current input record to the Teradata RDBMS either by itself or as the last of a sequence of concatenated records.
- **Note:** Regardless of whether the condition evaluates to true or false, the MultiLoad utility removes the tested string (the continuation indicator field) from each record.



.LAYOUT Parameters — CONTINUEIF

CONTINUEIF

- Record 1 and the immediately following Record 2 each represent a part of the input record.
- Continuation Character Field is defined beginning in character position 1.

Physical Input Records

1	2	10	45	80
Y	F1	F2	F3	

Record 1

1	2	35	80
N	FILLER	F5	

Record 2

MultiLoad .LAYOUT Definition

```
.LAYOUT Record_Layout CONTINUEIF 1= 'Y';
```

Logical Input Transaction

1	9	44	79 80	112 113	158
F1	F2	F3	FILLER	F5	

Note: F1 begins in character position 1 and Filler immediately follows F3.

.LAYOUT Parameters — INDICATORS

Use the INDICATORS parameters to handle nulls.

Miscellaneous Notes:

- When you use the INDICATORS specification, the MultiLoad utility sends *all of* the FIELD commands, including redefines, to the Teradata Database.
- **Caution:** Inappropriate INDICATORS specifications can corrupt the target table on the Teradata Database.
 - If INDICATORS is specified in the LAYOUT command and the data file does not contain indicator bytes in each record, the target table is loaded with incorrect data.
 - Conversely, if INDICATORS is not specified and the data file contains indicator bytes in each record, the target table also is corrupted.
- Always make sure that your INDICATORS specifications match the mode of the data you are sending to the Teradata Database.
- **Note:** INDICATORS processing is done only after any CONTINUEIF processing is completed for a record.



.LAYOUT Parameters — INDICATORS

INDICATORS

- The INDICATORS contain bits that, when equal to 1, represent a null data field.

Physical Input
Records

Indicator Byte(s)	F1	F2	F3

MultiLoad .LAYOUT
Definition

.LAYOUT Record_Layout INDICATORS;

.FIELD and .FILLER

When .FIELD is used, you specify a name for the field, the starting position or asterisk (*), the data type, and possibly options governing the handling of the input data.

By specifying a .FILLER, you define input data that will not be sent to the database table. A .FILLER statement requires a name, a starting position or asterisk, and the data type.

Performance Considerations

The majority of client processing during a MultiLoad job occurs when it is processing its input rows. The most efficient means of sending the row to the Teradata Database would be a bulk move of the input row to the output row.

However, there are many cases where fields need to be evaluated and field data may need to be individually moved from the input to the output row. Note, however, that performance is affected whenever a field needs to be evaluated or individually moved.

The need for moving individual field data from the input to the output row occurs for any of the following scenarios:

- DROP syntax on FIELD statements
- FILLER fields
- Concatenated fields
- Complex layout (first field is variable-length field, redefinition of field positions)

In addition to the above scenarios, variable length fields, NULLIF in the layout, and APPLY WHERE clauses might require additional CPU consumption.

If possible, try to avoid using the above options to improve MultiLoad performance.



.FIELD and .FILLER

```
.FIELD fieldname { startpos datadesc } || fieldexp  
[ NULLIF nullexpr ]  
[ DROP {LEADING / TRAILING } { BLANKS / NULLS }  
[ [ AND ] {TRAILING / LEADING } { NULLS / BLANKS } ] ] ;  
  
.FILLER [ fieldname ] startpos datadesc ;
```

.FIELD

- Input fields supporting redefinition and concatenation.

Startpos identifies the start of a field relative to 1.

Fieldexpr specifies a concatenation of fields in the format:

fieldname1 || fieldname2 [|| fieldname3 ...]

The option **DROP LEADING / TRAILING BLANKS / NULLS** is applicable only to character datatypes, and is sent as a VARCHAR with a 2-byte length field.

.FILLER

- Identifies data NOT to be sent to the Teradata database.

.LAYOUT Command — Examples

The facing page shows two examples using .LAYOUT: one with .TABLE and one with .FIELD and .FILLER.



.LAYOUT Command — Examples

Example 1:

```
.LAYOUT transrecord
CONTINUEIF 7 = 'ABC'
INDICATORS;

.FIELD field1 1 (char(5)) NULLIF field1 = 'AAAAA' DROP LEADING BLANKS ;
.FILLER field2 * (char(1));
.FIELD field3 * (char(3));
.FIELD field4           field2 || '&'amp; || field3;
```

Example 2:

```
.LAYOUT emp_record;
.TABLE Employee;
```

Note:

Employee is an existing table whose column names and data descriptions are assigned, in the order defined, to fields of the input data records.

Redefining the Input – Example

The input file may contain different types of records. Fields within these records start from the beginning of the record (position 1). Subsequent fields are indicated by an asterisk or by a starting character position.

By indicating the starting position of the field within the record, you can redefine the record. Remember, the asterisk (*) refers to the next position after the preceding field.

In the example on the following page, F4 starts in position 6 (right after the ‘.FILLER Type’ and ‘.FIELD PI’). F5 follows F4.



Redefining the Input — Example

- A bank loads daily transactions sequentially on a tape for batch processing by MultiLoad.
- An input data record might be an Add, Update or Delete, each of which has a different length and contains different fields, as illustrated:

Add New Account



Update Existing Account



Delete Inactive Account



```
.LAYOUT Record_Layout ;
.FILLER trans_type      1  CHAR(1) ;
.FIELD  PI              2  INTEGER ;
.FIELD  F1              *  INTEGER ;
.FIELD  F2              *  CHAR(25) ;
.FIELD  F3              *  CHAR(20) ;
.FIELD  F4              6  CHAR(2) ;
.FIELD  F5              *  INTEGER ;
```

Note:

FILLER data is
not placed into
ML work tables.

The .DML Command Options

The DML command provides labels for one or more DML statements (INSERT, UPDATE, or DELETE). The format of this command is shown next along with the options on this command.

You have already seen examples of how some of these options can be used to direct the processing of an UPSERT. The MARK option indicates that duplicate or missing rows should be recorded in one of the error tables.

MARK is the default for INSERT, UPDATE, and DELETE. If an error occurs in the Application Phase (e.g., uniqueness violation) with MARK specified, then the duplicate row is placed in to the UV_errtable. If a row is missing with an UPDATE or DELETE, then that transaction is also placed in the UV_errtable.

IGNORE is the default for UPSERT if the UPSERT is successful. If an UPSERT fails because of a constraint violation, the error is placed in the UV_errtable.

For import tasks, you can specify as many as five distinct error treatment options with one DML LABEL command. For example:

```
.DML LABEL COMPLEX
  IGNORE DUPLICATE INSERT ROWS
  MARK DUPLICATE UPDATE ROWS
  IGNORE MISSING UPDATE ROWS
  MARK MISSING DELETE ROWS
  DO INSERT FOR MISSING UPDATE ROWS;
```

Notes:

- If a uniqueness violation occurs with MARK specified, the duplicate rows go to the uniqueness violation table.
- IGNORE DUPLICATE ROWS does not apply if there are any unique indexes in the table.
- In the case of an *upsert* operation, both the insert and update portions must fail for an error to be recorded. In this case, the “marked” rows for the missing update operations then have nulls for the target table columns.
- If you do not specify either INSERT or UPDATE with DUPLICATE, then the MARK or IGNORE specification applies to both insert and update operations.
- Similarly, if you do not specify either UPDATE or DELETE with MISSING, then the MARK or IGNORE specification applies to both update and delete operations.
- MARK is the default for all actions except MISSING UPDATE for an *upsert* operation.



The .DML Command Options

Defines Labels along with Error Treatment conditions for one or more following INSERTs, UPDATEs or DELETEs to be applied under various conditions:

```
.DML LABEL Labelname
[ { MARK | IGNORE } DUPLICATE [ INSERT | UPDATE ] ROWS
{ MARK | IGNORE } MISSING [ UPDATE | DELETE ] ]
DO INSERT FOR [MISSING UPDATE] ROWS ] ;
```

MARK or IGNORE

Whether or not to record duplicate or missing INSERT, UPDATE, OR DELETE rows into the UV_error_table and continue processing.

Operation:

- INSERT (Duplicate violation)
- UPDATE (Duplicate row violation)
- UPDATE (Fails - missing row)
- DELETE (Fails - missing row)
- UPSERT (If successful)
- UPSERT (Fails)

Default:

- Marked in UV_tablename
- Marked in UV_tablename
- Marked in UV_tablename
- Marked in UV_tablename
- Ignored
- Mark failure of INSERT in UV_tablename

Example of UPSERT failure:

1. PI value doesn't exist, so UPDATE can't occur.
2. INSERT fails because of check violation - e.g., can't put character data in a numeric field.

The .DML Command Options (cont.)

The syntax of the .DML Label command is repeated on the facing page for your convenience.

The default for an UPSERT operation is to not mark missing update rows.

When the MARK MISSING UPDATE ROWS is used with an UPSERT, this will list (in the UV_errtable) data rows that can't be updated (the row doesn't exist with the PI value). If the insert also fails (e.g., constraint violation), the insert record is also marked in the UV_errtable. In this case, one record can cause 2 rows to go into the UV_errtable – one for the missing update and one for the insert failure.



The .DML Command Options (cont.)

```
.DML LABEL Labelname
[ { MARK | IGNORE } DUPLICATE [ { MARK | IGNORE } MISSING [ { INSERT | UPDATE } UPDATE | DELETE ] ROWS
DO INSERT FOR [ MISSING UPDATE ] ROWS ] ;
```

DO INSERT FOR MISSING UPDATE Key statement that indicates an UPSERT. An SQL UPDATE followed by an SQL INSERT is required.

Example 1: **.DML LABEL Action1 DO INSERT FOR MISSING UPDATE ROWS;**

The default for an UPSERT operation is to not mark missing update rows.

Example 1: **.DML LABEL Action2 MARK MISSING UPDATE ROWS
DO INSERT FOR MISSING UPDATE ROWS;**

When the MARK MISSING UPDATE ROWS is used with an UPSERT, this will place (in the UV_table) data rows that can't be updated (no PI). If the insert also fails, the insert record is also marked in the UV_table.

MultiLoad Statistics

Statistics indicating the number of records processed by each DML for *each table* is reported at the end of the Application Phase. The facing page shows an example of this output.



MultiLoad Statistics

At the end of the application phase, MultiLoad provides statistical information.

```
****06:06:41 UTY1803 Import Processing Statistics
      Total
      Import 1   Thus far
Candidate Records considered . . . . .    200000    200000
Apply conditions satisfied . . . . .    200000    200000

****06:18:34 UTY0818 Statistics for table ACCOUNTS:
  Inserts:      50000
  Updates:      50000
  Deletes:       0

****06:18:35 UTY0818 Statistics for table ACCOUNT_CUSTOMER:
  Inserts:      50000
  Updates:        0
  Deletes:       0

****06:18:35 UTY0818 Statistics for table CUSTOMER:
  Inserts:      25000
  Updates:      25000
  Deletes:       0

****06:18:35 UTY0818 Statistics for table TRANS_HIST:
  Inserts:      50000
  Updates:        0
  Deletes:       0
```

INMOD

An INMOD is an exit routine that can precondition data and pass it on to the loader. You can write INMODs to pre-screen the input data being sourced into MultiLoad.

INMOD and MultiLoad use a return code value to communicate with each other. You can write INMODs as restartable routines so that they can synchronize with the loader's checkpoints.

When an INMOD-connected loader restarts, both the utility and the INMOD can be repositioned to the last checkpoint.

Use INMODs to perform unusual conversions of data, for example, adding a sequenced column to the data, or reading data from a non-standard input file format.

Other IMPORT options

There are other options for the IMPORT command, including the specification of an AXSMOD.

AXSMOD

AXSMOD is used to specify an access module file that imports data from a file.

The AXSMOD option is not required for importing:

- Disk files on either network- or channel-attached systems
- Magnetic tape files on channel-attached client systems.

It is *required* for importing magnetic tape and other types of files on *network-attached* client systems.



INMODs



INMOD to MultiLoad return codes: After a READ call.

0	Valid data record in Buffer — EOF not reached. Length field reflects correct length of output record. If an input record was supplied to MLOAD from the INMOD and is to be skipped, the length field should be set to zero. If no input record was supplied, setting the length to zero indicates EOF.
Non 0	INMOD indicates end-of-file condition.

INMOD to MultiLoad return codes: After a non-READ call.

0	Indicates a successful operation.
1	Indicates a processing error; MLOAD will terminate

MultiLoad to INMOD additional return codes.

0	Calling for the first time. INMOD should open files to read data. MLOAD expects a record.
1	MultiLoad expects a record.
2	MLOAD has been restarted. INMOD should position itself to the last checkpoint. There may be re-positioning information in the data buffer.
3	Request for INMOD to take a Checkpoint. The INMOD should return any information (up to 100 bytes) needed to reposition itself. MLOAD saves this in the Restart Log Table.
4	Request for INMOD to reposition itself at the last checkpoint. Repositioning information may be in the data buffer.
5	Request for INMOD to wrap up at termination.
6	Request for INMOD to initialize and receive record.
7	Request for INMOD to receive next record.

Note:

MultiLoad can also use the FastLoad INMOD return codes.

Summary

The facing page summarizes some of the important concepts regarding the MultiLoad utility.



Summary

- On the .BEGIN statement, optionally, the names of work and error tables can be specified.
- You can:
 - Specify error limits and checkpoints.
 - Limit sessions.
 - Designate time allowed for connection.
 - Specify retry time limit.
 - Designate the level of notification you prefer.
 - Designate how MultiLoad will proceed if AMPs are offline.
- .LAYOUT parameters define the data format.
- .DML commands define Labels and Error Treatment conditions for one or more operations.
- You can use FastLoad or MultiLoad INMODs.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. Complete the BEGIN statement to accomplish the following:

- Specify an error limit count of 200,000 and an error percentage of 5%.
- Specify a checkpoint at 500,000 records.
- Request 16 sessions, but allow the job to run with only 8.
- Set the number of hours to try to establish connection as 6.

```
.LOGTABLE RestartLog_mld;  
.LOGON _____;  
.BEGIN [IMPORT] MLOAD TABLES Trans_Hist  
  
;  
.END MLOAD ;
```

Lab Exercise 38-1

The size of data38_1 should be 26,700 bytes.

The execution of the macro AP.Lab7_1 will cause 300 rows to be created in the data38_1 file. 100 of these records will start with an A, 100 will start with a C, and 100 will start with a T.

The format of data38_1 is:

A	Accounts record
C	Customer record
T	Trans record

The control letter (A, C, or T) must be in upper-case letters in the APPLY statements. The formats of the Accounts, Customer, and Trans records are defined in Appendix A.

Important note:

The Trans_Date is output as a date in ANSI date ('YYYY-MM-DD') format of CHAR(10).

Therefore, define the date as CHAR(10) for input.

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab38_12.mld** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function
To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 38-1

Purpose

In this lab, you will use MultiLoad to insert the data rows you deleted in a lab from Module 37. The Accounts, Trans, and Customer tables each should have 100 rows in them (from Lab 37-1).

What you need

Populated tables and macro AP.Lab38_1.

Tasks

1. Export data to a file called *data38_1* by executing the macro AP.Lab38_1. Submit the following commands in BTEQ:

```
.EXPORT DATA FILE = data38_1;  
EXEC AP.LAB38_1;  
.EXPORT RESET;
```

Note: The *Trans_Date* is exported with an ANSI Date Format of 'YYYY-MM-DD' and a data type of CHAR(10).

2. Prepare a MultiLoad script which inserts rows into one of three different tables using the redefinition feature of MultiLoad. The input record contains a mixture of records which are to be inserted into the Accounts table if the first record byte is an "A", the Customer table if the first byte is a "C" and the Trans table if the first byte is a "T".
3. Check your tables for 200 rows per table. Your MultiLoad job should have a final return code of zero and should evidence 100 rows inserted into each of the three tables.

Lab Exercise 38-2

The size of data38_2 should be 15,600 bytes.

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab38_23.mld** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function

To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 38-2

Purpose

In this lab, you will prepare and execute a MultiLoad script which performs an 'UPSERT' operation (INSERT MISSING UPDATE) on your Accounts table as a single operation.

What you need

Populated tables and macro AP.Lab38_2.

Tasks

1. Delete all rows from the Accounts Table and use the following INSERT/SELECT to create 100 rows of data in your table.

```
INSERT INTO Accounts SELECT * FROM AP.Accounts WHERE Account_Number LT 20024101 ;
```

2. Export data to a file *data38_2* using the macro AP.Lab38_2.

```
.EXPORT DATA FILE = data38_2 ;  
EXEC AP.Lab38_2 ;  
.EXPORT RESET ;
```

3. Prepare and execute a MultiLoad script which performs an 'UPSERT' operation (INSERT MISSING UPDATE) on your Accounts table as a single operation. Use the data from *data38_2* as input to the MultiLoad 'UPSERT' script. If the row exists, UPDATE the Balance_Current with the appropriate incoming value. If not, INSERT a row into the Accounts table.
4. Validate your results. MultiLoad should have performed 100 UPDATES and 100 INSERTS with a final return code of zero.

Teradata Training

Notes

Module 39



TPump

After completing this module, you will be able to:

- State the capabilities and limitations of TPump.
- Describe TPump commands and parameters.
- Prepare a TPump script.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

TPump	39-4
ATOMIC UPSERT	39-4
TPump Limitations	39-6
.BEGIN LOAD Statement	39-8
Notes on CHECKPOINT	39-8
Notes on ERRLIMIT	39-8
TPump Specific Parameters	39-10
SERIALIZE	39-10
PACK	39-10
RATE	39-10
LATENCY	39-10
NOMONITOR	39-10
ROBUST	39-10
MACRODB	39-10
.BEGIN LOAD – PACK and RATE	39-12
.BEGIN LOAD – SERIALIZE OFF	39-14
.BEGIN LOAD – SERIALIZE ON	39-16
Latency	39-16
.BEGIN LOAD – ROBUST ON	39-18
Sample TPump Script (1 of 2)	39-20
Sample TPump Script (2 of 2)	39-22
TPump Compared with MultiLoad	39-24
Economy of Scale and Performance	39-24
Multiple Statement Requests	39-24
Macro Creation	39-24
Locking and Transactional Logic	39-24
Additional TPump Statements	39-26
DATABASE	39-26
EXEC(UTE)	39-26
Invoking TPump	39-28
TPump Statistics	39-30
TPump Monitor	39-32
Checking Status of a Job	39-32
Changing the Statement Rate as a Job Runs	39-32
INMOD	39-34
Application Utility Checklist	39-36
Summary	39-38
Review Questions	39-40
Lab Exercise 39-1	39-42

TPump

TPump (**Teradata Parallel Data Pump**) provides an excellent utility for low-volume batch maintenance of large Teradata databases. It enables acquisition of data from the client with low processor utilization. TPump is as flexible as BulkLoad (an older Teradata utility that is no longer supported), which it has replaced.

The functionality of TPump is enhanced by the Support Environment. In addition to coordinating activities involved in TPump tasks, it provides facilities for managing file acquisition, conditional processing, and certain DML (Data Manipulation Language) and DDL (Data Definition Language) activities on the Teradata Database. The Support Environment enables an additional level of user control over TPump.

TPump uses row-hash locks, making concurrent updates on the same table a possibility.

TPump has a built-in resource governing facility that allows the operator to specify how many updates occur (the statement rate) minute by minute, and then change the statement rate while the job continues running. This utility can be used to increase the statement rate during windows when TPump is running by itself, and then decrease the statement rate later on if users log on for ad-hoc query access.

TPump can always be stopped and all its locks can be dropped with no ill effect.

The facing page identifies the principal features of the TPump utility.

ATOMIC UPSERT

TPump has support for ATOMIC UPSERT. This enhances active warehouse transactions by allowing TPump to perform PACKed UPSERT operations without the cost of rollbacks that were incurred in previous UPDATE then INSERT transactions. UPSERT is a composite of UPDATE and INSERT operations.

The one-pass UPSERT is called atomic to emphasize that both the UPDATE and the INSERT are grouped together and executed as a single SQL statement. The syntax has been modified to allow an optional ELSE in the UPDATE statement. ATOMIC UPSERT makes inserting faster because it requires only one lock, no one can change the table during the ATOMIC UPSERT, and the client application sends and receives fewer packets during inserts, which improves performance.



TPump

- Allows near real-time updates from transactional systems into the warehouse.
- Performs **INSERT, UPDATE, and DELETE operations, or a combination, from the same source. Up to 128 DML statements can be included for one IMPORT task.**
- Alternative to MultiLoad for low-volume batch maintenance of large tables.
- Allows target tables to:
 - Have secondary indexes, join indexes, hash indexes, and Referential Integrity.
 - Be MULTISET or SET.
 - Be populated or empty.
 - Tables can have triggers - invoked as necessary
- Allows conditional processing (via APPLY in the .IMPORT statement).
- Supports automatic restarts; uses Support Environment.
- No session limit — use as many sessions as necessary.
- No limit to the number of concurrent instances.
- **Uses row-hash locks, allowing concurrent updates on the same table.**
- Can always be stopped and locks dropped with no ill effect.
- Designed for highest possible throughput.
 - User can specify how many updates occur minute by minute; can be changed as the job runs.

TPump Limitations

The facing page lists some TPump limitations you should be aware of.

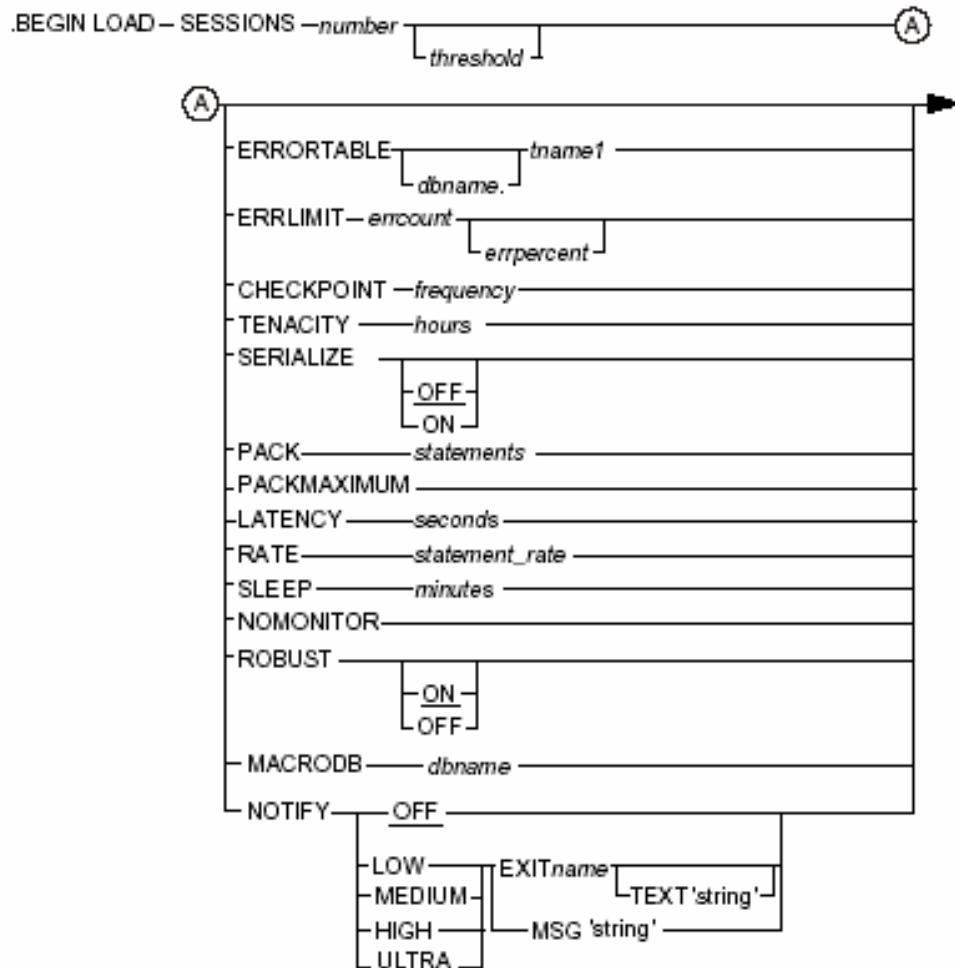


TPump Limitations

- Use of **SELECT** is not allowed.
- Concatenation of data files is not supported.
- Exponential operators are not allowed.
- Aggregate operators are not allowed.
- Arithmetic functions are not supported.
- There is a limit of four **IMPORT** commands within a single TPump "load" task.
- In using TPump with dates before 1900 or after 1999, the year portion of the date must be represented by four numerals (yyyy).
 - The default of two numerals (yy) to represent the year is interpreted to be the 20th century.
 - The correct date format must be specified at the time of table creation.

.BEGIN LOAD Statement

Note that many of the parameters are similar to those used in the MultiLoad utility. The format of the .BEGIN LOAD statement is:



Notes on CHECKPOINT

Note that for TPump, only the *frequency* option is used for checkpoints. MultiLoad also allows number of records. If you specify a CHECKPOINT *frequency* of more than 60, TPump terminates the job. Specifying a CHECKPOINT *frequency* of zero bypasses the check pointing function.

Notes on ERRLIMIT

In extreme cases (all records have errors), if the number of statements in each request (PACK factor) is smaller than the ERRLIMIT, the job can stop due to exceeding the ERRLIMIT, producing no error table rows. To avoid this, set the ERRLIMIT greater than the PACK factor.



.BEGIN LOAD Statement

Many of the .BEGIN parameters are comparable to those for MultiLoad.

.BEGIN LOAD		
SESSIONS	<i>max [min]</i>	(required)
ERRORTABLE	<i>tablename</i>	(defaults to <i>jobname_ET</i>)
ERRLIMIT	<i>errcount</i>	[<i>errpercent</i>]
CHECKPOINT	<i>frequency</i>	(default is 15 minutes)
TENACITY	<i>hours</i>	(default is 4)
SLEEP	<i>minutes</i>	(default is 6)

However, TPump has numerous parameters on the .BEGIN LOAD statement that are unique to TPump.

SERIALIZE	<u>ON OFF</u>	(default ON if UPSERT)
PACK	<i>number</i>	(default is 20, max is 600)
PACKMAXIMUM		(use maximum pack factor)
RATE	<i>number</i>	(default is unlimited)
LATENCY	<i>number</i>	(range is 10 – 600 seconds)
NOMONITOR		(default is monitoring on)
ROBUST	<u>ON OFF</u>	(default is ON)
MACRODB	<i>dbname</i>	(default is logtable dbase) ;

TPump Specific Parameters

There are a number of parameters specific to TPump.

SERIALIZE

If ON, this option guarantees that operations on a row occur serially. If SERIALIZE is specified without ON or OFF, the default is ON. If SERIALIZE is not specified, the default is OFF unless the job contains an UPSERT operation which causes SERIALIZE to default to ON. This feature is meaningful only when a primary index for the table is specified by using the KEY option with the FIELD command.

Ex. .FIELD ACCTNUM * INTEGER KEY;

PACK

Specifies the number of statements to pack into a multiple-statement request. Packing improves network/channel efficiency by reducing the number of sends and receives between the application and Teradata.

RATE

Specifies the initial maximum rate at which statements are sent to the Teradata RDBMS per minute. If the statement rate is either zero or unspecified, the rate is unlimited. If the statement rate is less than the statement packing factor, TPump sends requests smaller than the packing factor. If the TPump monitor is in use, the statement rate can be changed later.

LATENCY

Allows TPump to commit to Teradata any data sitting in the buffer longer than the LATENCY value. Allows TPump to become a multi-threaded application in the event that control of the main thread is awaiting input from an access module. The range is 10-600 seconds.

NOMONITOR

Prevents TPump from checking for statement rate changes from or update status information for the TPump Monitor application.

ROBUST

The OFF parameter signals TPump to use “simple” restart logic. In this case, restarts cause TPump to begin where the last checkpoint occurred in the job. Any processing that occurred after the checkpoint is redone. This method does not have the extra overhead of additional database writes in the robust logic. Note, that certain errors may cause reprocessing, resulting in extra error table rows due to re-executing statements (attempting to re-insert rows) which previously resulted in the errors.

In Robust Mode, Teradata writes 1 database row in the Restart Log table for every SQL transaction.

MACRODB

Specifies the database to contain any macros used by TPump. If not specified, the default database that TPump uses to place create macros is the same database as the Restart Log table.



TPump Specific Parameters

Specific TPump .BEGIN LOAD parameters are:

SERIALIZE	<u>ON / OFF</u>	ON guarantees that operations on a given key combination (row) occur serially. Used only when a primary index is specified. KEY option must be specified if SERIALIZE ON.
PACK	statements	Number of statements to pack into a multiple-statement request.
PACKMAXIMUM		Number of statements to pack into a multiple-statement request.
RATE	statement rate	Initial maximum rate at which statements are sent per minute. If the statement rate is zero or unspecified, the rate is unlimited.
LATENCY	seconds	# of seconds before a partial buffer is sent to the database.
NOMONITOR		Prevents TPump from checking for statement rate changes from or update status information for the TPump Monitor.
ROBUST	<u>ON / OFF</u>	OFF signals TPump to use “simple” restart logic; TPump will begin where the last checkpoint occurred.
MACRODB	dbname	Indicate a database to contain any macros used by TPump.

.BEGIN LOAD – PACK and RATE

PACK specifies the number of statements to pack into a multi-statement request. This improves network/channel efficiency by reducing the number of sends and receives between the application and Teradata. Up to a maximum of 600 statements may be specified. If the TPump parser issues a warning that it has reduced the requested packing rate to a value, change your script to this value. This will reduce the overhead caused by dynamic adjusting of TPump.

The Teradata Database enforces a maximum column limit with TPump jobs. This limit was raised with TTU7.0/V2R5.0 from 507 to 2560.

Example – if you issue a TPump job as a multi-statement request, with a USING clause that has 64 columns, you could divide 2560 by 64 to give you the maximum PACK factor you could use – but note that this will most likely not provide the best performance level, and will most certainly blow out the (unknown) plastic steps limit. The other restrictions that must be considered are:

- 64K message size limit
- TPump limit of 600 statements
- Teradata USING limit of 2560 columns
- Plastic Steps limit

In order to determine the best PACK rate for a TPump job, you need to experiment with various numbers. As you increase the PACK rate, the throughput improvement is great at first, then falls off and gets worse. Going from a PACK rate of 1 to 2 could provide huge performance gains, and going from 2 – 4 could be just as beneficial, but moving from 8 to 16 might cause a performance drop. You could run tests at 2, 4, 8, 16, 32 and 64 and graph the results. One goal is to find the “sweet spot” that provides the best performance for your job, and another goal is to find the maximum PACK rate that your job can use.

PACKMAXIMUM can be used to determine what the MAX PACK can be. Do not run PACKMAXIMUM against production jobs, because it determines the PACK factor by trial and error. Do this once, in a test situation, to determine the maximum PACK factor you could employ.

NOTE: Clean Data

If your data has errors in it, larger packs can hurt – this is because of the way TPump handle errors. If you have a few hundred statements and an error occurs, Teradata rolls back the entire request, and TPump has to remove the statement and the data, and then reissue the entire statement. It is very important to have CLEAN data when using TPump.



.BEGIN LOAD – PACK and RATE

- **PACK** specifies the number of statements to pack into a multi-statement request.
 - Improves network/channel efficiency by reducing the number of sends and receives between the application and Teradata.
 - Increasing the PACK rate improves throughput performance – to a certain level.
- **PACKMAXIMUM** – use in testing to help establish a PACK value; do not use in a production job.
- **Restrictions to consider:**
 - 64K message size limit
 - TPump limit of 600 statements
 - Teradata USING clause limit of 2560 columns (from 507)
 - Teradata Plastic Steps limit
- **RATE** sets the initial maximum rate at which statements are sent per minute.
 - Defaults to unlimited; specify a value to control the number of amount of work sent to Teradata.
 - Example: RATE 12000 PACK 20
 $12000 \div 20 = 600$ packets/minute or approximately 10 packets/second

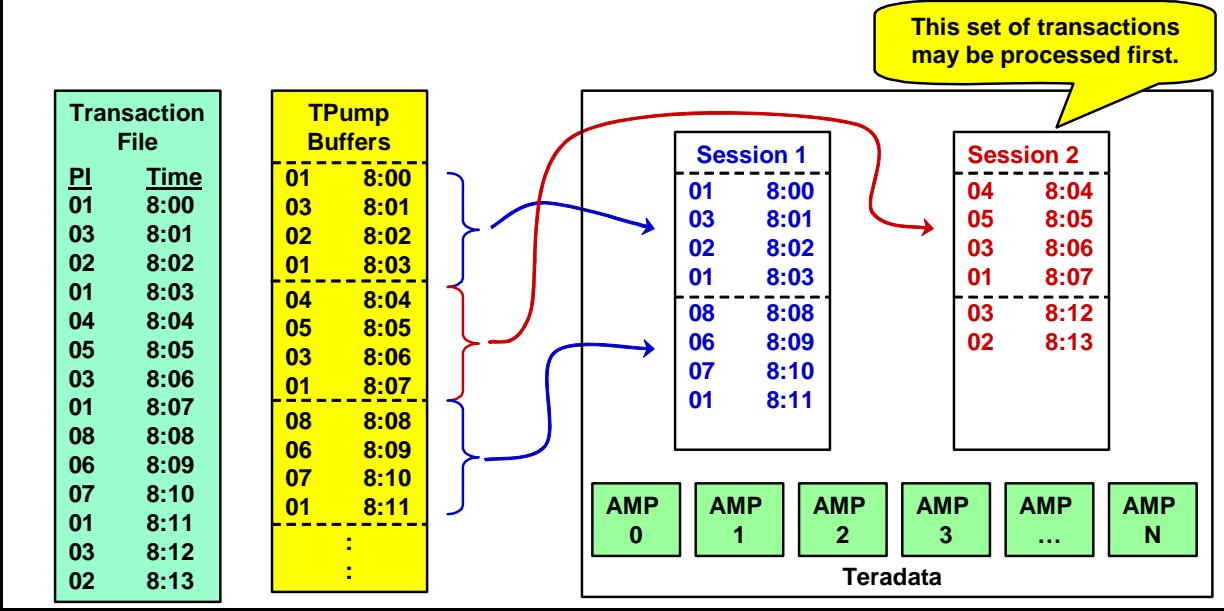
.BEGIN LOAD – SERIALIZED OFF

With SERIALIZED OFF, statements are executed on the first session available; hence, operations may occur out of order.

The example on the facing page illustrates how transactions are processed by TPump.

.BEGIN LOAD – SERIALIZE OFF

- With SERIALIZE OFF, transactions are processed in the order they are encountered and placed in the first available buffer. Buffers are sent to PE sessions and different PEs process the data independently of other PEs.
- SERIALIZE OFF does not guarantee the order in which transactions are processed.**



.BEGIN LOAD – SERIALIZED ON

SERIALIZED ON tells TPump to partition the input records across the number of sessions it is using, ensuring that all input records that touch a given target table row (or that contain the same non-unique primary index value, for example) are handled by the same session.

SERIALIZED ON is useful for two reasons:

1. The order that the updates are applied is important in this application.
2. There is a possibility that rows with the same primary index value will be inserted through different buffers at the same time.

This second point has performance implications, as SERIALIZED ON can eliminate the lock delays or potential deadlocks caused by primary index collisions.

This guarantees both input record order and all the records with the same primary index value will be handled in the same session, and possibly the same buffer. The way SERIALIZED guarantees input order is to partition on the columns you have specified in your TPump script as KEY fields. Usually this will be the primary index of the table being updated, but it may be a different column or set of columns. It could, for example, be the primary index column(s) of a join index built upon the table being loaded.

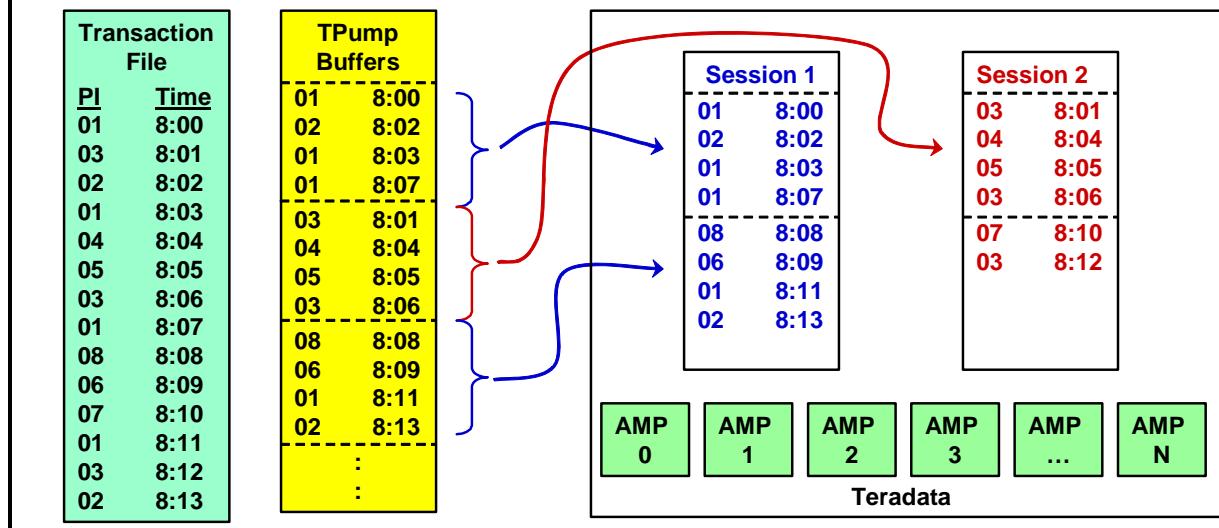
SERIALIZED reduces row blocking which could lead to deadlocks between buffers within the same TPump job, when rows with non-unique primary index values are being processed. Manual partitioning is required to do the same if the input data is divided between multiple TPump jobs.

Latency

Latency (seconds) is the number of seconds to use as a threshold for flushing stale buffers, based on the number of seconds the oldest record is in the buffer. The range is from 10 to 600 seconds. If serialization is off, only the current buffer can be stale. If serialization is on, the number of stale buffers can range from zero to the number of sessions.

.BEGIN LOAD – SERIALIZE ON

- SERIALIZE ON can eliminate lock delays or potential deadlocks caused by primary index collisions, improving performance.
- SERIALIZE guarantees both input record order and all records with the same PI value will be handled in the same session. It is recommended to specify the PI in the statement column(s) as KEY.
- KEY Fields determine the PE session in which TPump send the transaction to.



.BEGIN LOAD – ROBUST ON

ROBUST ON is the default for all TPump jobs. By inserting some extra information about the rows just processed into the database log table at the completion of each buffer's request, TPump has a way to avoid re-applying rows that have already been processed in the event of a restart.

The ROBUST ON variable causes a row to be written to the log table each time a buffer has successfully completed its updates. These mini-checkpoints are deleted from the log when a checkpoint is taken, and are used during a restart to identify which rows have already been successfully processed since the most recent checkpoint was taken, then bypassing them on a restart. The larger the TPump pack factor, the less overhead involved in this activity.

ROBUST ON is particularly important if re-applying rows after a restart would cause either a data integrity problem or have an unacceptable performance impact. ROBUST ON is advisable for these specific conditions:

1. INSERTs into multi-set tables, as such tables will allow reinsertion of the same rows multiple times.
2. When updates are based on calculations or percentage increases
3. If pack factors are large, and applying and rejecting duplicates after a restart would be unduly time-consuming.
4. If data is time-stamped at the time it is inserted into the database.

ROBUST ON is always a good idea for TPump jobs that read from queues, but is particularly important if timestamps are used to record the time of insertion into the database. The original rows that were inserted before the restart will carry a timestamp that reflects their insertion time. If a reapply of a row is attempted, the reapplied row will carry a timestamp that is different, even though all of the other data is identical. To Teradata, this will appear as a different row with the same primary index value, so duplicate row checking will not prevent the duplicate insertion. ROBUST ON is needed to keep duplicates from being added to the table being loaded in the case of restart.



.BEGIN LOAD – ROBUST ON

- ROBUST ON is the default for all TPUMP jobs.
- This option avoids re-applying rows that have already been processed in the event of a restart.
- Causes a row to be written to the log table each time a buffer has successfully completed its updates.
 - The larger the TPump PACK factor, the less overhead involved in this activity.
- These rows are deleted from the log when a checkpoint is taken.
- ROBUST ON is recommended for these specific conditions:
 - INSERTS into multi-set tables, as such tables will allow re-insertion of the same rows multiple times.
 - When UPDATEs are based on calculations or percentage increases.
 - If PACK factors are large, and applying and rejecting duplicates after a restart would be time-consuming.
 - If data is time-stamped at the time it is inserted into the database.
- ROBUST ON is always a good idea for TPump jobs that read from queues. It keeps duplicates from being re-inserted into the table in the event of a restart.

Sample TPump Script (1 of 2)

The next few pages provide an example of a TPump script.

Miscellaneous TPump Notes:

- With TPump, all required data is imported; none is obtained from tables already in the Teradata Database.
- No statement of an IMPORT task may make any reference to a table or row other than the one affected by the statement.
- TPump rejects UPDATEs that try to change to Primary Index value.
- .DML UPDATE requires a WHERE clause.
- .DML DELETE must not contain any joins
- .DML DELETE cannot have an OR (alternative is to use 2 separate .DML DELETE statements)
- MARK is default for INSERT, UPDATE, and UPDATE. IGNORE is the default for UPSERT.
- DUPLICATE – if a duplicate row is created as a result of an UPDATE or an INSERT (and the table doesn't support duplicate rows), the duplicate row is either ignored (IGNORE) or placed (MARK) in the Error_Table.
- MISSING – if a row is missing on an UPDATE or DELETE, the transaction is either ignored (IGNORE) or placed (MARK) in the Error_Table.
- EXTRA – lets you know if multiple rows are affected. If duplicate rows already exist and an UPDATE or a DELETE impacts multiple duplicate rows, then either the duplicates are ignored (IGNORE) or placed (MARK) in the Error_Table.



Sample TPump Script (1 of 2)

```
.LOGTABLE restartlog39_tpp;
.LOGON tdpid/username,password;
.BEGIN LOAD      SESSIONS 8          SERIALIZE ON
                  PACK 20           RATE 12000
                  ERRORTABLE ACT_tpp_ET  ERRLIMIT 100 ;
.LAYOUT layout12;
.FIELD table_code          1      CHAR(1);
.FIELD A_accountno         2      INTEGER          KEY;
.FIELD A_number             *      INTEGER;
.FIELD A_street             *      CHAR(25);
.FIELD A_city               *      CHAR(20);
.FIELD A_state              *      CHAR(2);
.FIELD A_zipcode            *      INTEGER;
.FIELD A_balancefor         *      DECIMAL(10,2);
.FIELD A_balancecur         *      DECIMAL (10,2);
.FIELD C_customer_number    2      INTEGER          KEY;
.FIELD C_last_name          *      CHAR(30);
.FIELD C_first_name         *      CHAR(20);
.FIELD C_social_security   *      INTEGER;
.FIELD T_trans_number       2      INTEGER;
.FIELD T_trans_date         *      CHAR(10);
.FIELD T_accountno          *      INTEGER          KEY;
.FIELD T_trans_id            *      CHAR(4);
.FIELD T_trans_amount        *      DECIMAL(10,2);
```

Sample TPump Script (2 of 2)

The facing page shows the rest of the example TPump script. Note the two IMPORT clauses.

Note about the USE option with .DML LABEL. TPump uses all of the fields in the layout in 1) the macro definition, 2) the using clause of the macro definition, and 3) sends them in the data parcel even if they aren't referenced in the values clause.

- To minimize the amount of data placed into a data parcel, use the USE option to only specify the fields needed for the SQL statements that are part of the .DML LABEL.
- A problem can also occur when a field is redefined over another field with an incompatible data type. Use of the USE option helps avoid this problem.



Sample TPump Script (2 of 2)

```
.DML LABEL Ins_Account
USE (A_accountno, A_number, A_street, A_city, A_state, A_zipcode, A_balancefor, A_balancecur);
INSERT INTO Accounts VALUES
(:A_accountno, :A_number, :A_street, :A_city, :A_state, :a_zipcode, :A_balancefor, :A_balancecur);

.DML LABEL Ins_Customer
USE (C_customer_number, C_last_name, C_first_name, C_social_security);
INSERT INTO Customer VALUES
(:C_customer_number, :C_last_name, :C_first_name, :C_social_security);

.DML LABEL Ins_Trans
USE (T_trans_number, T_trans_date, T_accountno, T_trans_Id, T_trans_amount);
INSERT INTO Trans VALUES
(:T_trans_number, :T_trans_date, :T_accountno, :T_trans_Id, :T_trans_amount);

.IMPORT INFILE datafile1 LAYOUT layout12
APPLY Ins_Account      WHERE table_code = 'A'
APPLY Ins_Trans         WHERE table_code = 'T'
APPLY Ins_Customer      WHERE table_code = 'C';

.IMPORT INFILE datafile2 LAYOUT layout12
APPLY Ins_Account      WHERE table_code = 'A'
APPLY Ins_Trans         WHERE table_code = 'T'
APPLY Ins_Customer      WHERE table_code = 'C';

.END LOAD;
.LOGOFF;
```

TPump Compared with MultiLoad

If you have both MultiLoad and TPump utilities, you may want to know which to use when. In fact, TPump compliments MultiLoad.

Economy of Scale and Performance

MultiLoad performance improves as the volume of changes increases. In phase two of MultiLoad, changes are applied to the target table(s) in a single pass and all changes for any physical data block are effected using one read and one write of the block. The temporary table and the sorting process used by MultiLoad are additional overhead that must be “amortized” through the volume of changes. TPump, on the other hand, does better on relatively low volumes of changes because there is no temporary table overhead. TPump becomes expensive for large volumes of data because multiple updates to a physical data block will most likely result in multiple reads and writes of the block.

Multiple Statement Requests

The most important technique used by TPump to improve performance is the use of a multiple statement request. Placing more statements in a single request is beneficial for two reasons. First, because it reduces network overhead since large messages are more efficient than small ones. Second, (in ROBUST mode) it reduces TPump recovery overhead which amounts to one extra database row written for each request. TPump automatically packs multiple statements into a request based upon the PACK specification in the BEGIN LOAD command.

Macro Creation

For efficiency, TPump uses macros to modify tables, rather than the actual DML commands. The technique of changing statements into equivalent macros before beginning the job greatly improves performance.

- The size of network (and channel) messages sent to the Teradata Database by TPump is reduced.
- Teradata Database parsing engine overhead is reduced because the execution plans (or “steps”) for macros are cached and re-used.

Locking and Transactional Logic

In contrast to MultiLoad, TPump uses row hash locking to allow for some amount of concurrent read and write access to its target tables. At any point TPump can be stopped and target tables are fully accessible. If TPump is stopped, however, depending on the nature of the update process, it may mean that the “relational” integrity of the data is impaired.

This differs from MultiLoad, which operates as a single logical update to one or more target tables. Once MultiLoad goes into phase two of its logic, the job is “essentially” irreversible and the entire set of table(s) is locked for write access until it completes. If TPump operates on rows that have associated “triggers,” the triggers are invoked as necessary.



TPump Compared with MultiLoad

- MultiLoad performance improves as the volume of changes increases.
- TPump does better on relatively low volumes of changes.



- TPump improves performance via a multiple statement request.



- TPump uses macros to modify tables rather than the actual DML commands.

Example of macro name – M2000216_105642_01_0001

- MultiLoad uses the DML statements.



- TPump uses row hash locking to allow for concurrent read and write access to target tables. It can be stopped with target tables fully accessible.
- In Phase 4, MultiLoad locks tables for write access until it completes.

Additional TPump Statements

The facing page identifies additional statements used by TPump.

DATABASE

The DATABASE statement changes the default database qualification for all unqualified DML and DDL statements. It only affects “native SQL” commands, and has no effect on the BEGIN LOAD command. The DATABASE command does affect INSERT, UPDATE, DELETE and EXEC statements issued as part of a load. (When TPump logs on sessions, it immediately issues a DATABASE statement on each session.)

The DATABASE command does not affect the placement of TPump macros.

EXEC(UTE)

The EXECUTE statement specifies a user-created macro for execution. The macro named in this statement is resident in the Teradata RDBMS and specifies the type of DML statement (INSERT, UPDATE, DELETE, or UPSERT) being handled by the macro.

The EXECUTE command immediately follows .DML LABEL:

Rules for user-created macros include:

- TPump expects the parameter list for any macro to exactly match the FIELD list specified by the LAYOUT in the script. FILLER fields are ignored. If the USE clause is used in the DML statement, TPump expects the parameter list for every macro in the DML statement to exactly match the field list specified by the USE clause.
- The macro should specify a single prime index operation: INSERT, UPDATE, DELETE, or UPSERT. TPump reports an error if the macro contains more than one supported statement. If the EXECUTE statement is replacing an INSERT, UPDATE, DELETE, or UPSERT statement in a job script, the EXECUTE statement must be placed at the same location as the INSERT, UPDATE, DELETE, or UPSERT statement that it replaces.



Additional TPump Statements

DATABASE Changes the default database qualification for all DML statements.

EXEC(UTE) Specifies a user-created macro for execution. The macro named is resident in the Teradata database.

DATABASE database ;

EXECUTE [database.]macro_name {
 UPDATE/UPD
 INSERT/INS
 DELETE/DEL
 UPSERT/UPS} ;

Commands and statements in common with MultiLoad:

ACCEPT	IMPORT	RUN
DELETE	INSERT	SET
DISPLAY	LAYOUT	SYSTEM
DML	LOGON	TABLE
FIELD	LOGOFF	UPDATE
FILLER	LOG	
IF / ELSE / ENDIF	ROUTE	

Invoking TPump

The facing page displays the commands you can use to execute TPump.



Invoking TPump

Network Attached Systems: tpump [PARAMETERS] < *scriptname* >*outfilename*

Channel-Attached MVS Systems: // EXEC TDSTPUMP PARM= [PARAMETERS]

Channel-Attached VM Systems: EXEC TPUMP [PARAMETERS]

Channel Parameter	Network Parameter	Description
BRIEF	-b	Reduces print output runtime to the least information required to determine success or failure.
CHARSET= <i>charsetname</i>	-c <i>charsetname</i>	Specify a character set or its code. Examples are EBCDIC, ASCII, or Kanji sets.
ERRLOG= <i>filename</i>	-e <i>filename</i>	Alternate file specification for error messages; produces a duplicate record.
" <i>tpump command</i> "	-r ' <i>tpump cmd</i> '	Signifies the start of a TPump job; usually a RUN FILE command that specifies the script file.
MACROS	-m	Keep macros that were created during the job run.
VERBOSE	-v	Additional statistical data in addition to the regular statistics.
	< <i>scriptname</i>	Name of file that contains TPump commands and SQL statements.
	> <i>outfilename</i>	Name of output file for TPump messages.

TPump Statistics

For each task, TPump accumulates statistical items and writes them to the customary output destination of the external system, SYSPRINT/stdout (or the redirected stdout), or the destination specified in the ROUTE command.

Candidate records considered. The number of records read.

Apply conditions satisfied. Represents the number of statements sent to the RDBMS.

If there are no rejected or skipped records, this value is equal to the number of candidate records, multiplied by the number of APPLY statements referenced in the import.

Candidate records rejected. Represents the number of records that are rejected by the TPump client code because they are formatted incorrectly.

Candidate records with data errors (not applied). Represents the number of records resulting in errors on the Teradata Database. These records are found in the associated error table.

Statistics for Apply Label. This area breaks out the total activity count for each statement within each DML APPLY clause. The ‘Type’ column contains the values U for update, I for insert and D for delete. Note that unlike the other reported statistics, these values are NOT accumulated across multiple imports.



TPump Statistics

	IMPORT 1	Total thus far
Candidate records considered:.....	200	200
Apply conditions satisfied:.....	200	200
Candidate records not applied:.....	0	0
Candidate records rejected:.....	0	0

**** Statistics for Apply Label : UPS_ACCOUNT**

Type	Database	Table or Macro Name	Activity
U	TLJC25	Accounts	100
I	TLJC25	Accounts	100

**** 19:33:50 UTY0821 Error table TLJC25.errtable_tpp is EMPTY, dropping table.

0018 .LOGOFF;

```
=====
= = =
= Logoff/Disconnect = =
= = =
=====
```

**** 19:34:18 UTY6216 The restart log table has been dropped.

**** 19:34:18 UTY6212 A successful disconnect was made from the RDBMS.

**** 19:34:18 UTY2410 Total processor time used = '2.43 Seconds'

- . Start : 19:33:33 - TUE MAY 11, 2004
- . End : 19:34:18 - TUE MAY 11, 2004
- . Highest return code encountered = '0'.

Note: These statistics are not for the example TPump job shown earlier in this module.

TPump Monitor

The TPump Monitor facility provides run-time monitoring of the TPump job. It allows users, via a command line interface, to track and alter the rate at which requests are issued to the Teradata Database.

To install the TPump tables, views, and macros for the TPump monitor facility, modify the following UNIX script (with DBC password) and execute it with BTEQ.

/usr/etc/tpump_examples/tpumpar.csq

In Windows, the script to execute with BTEQ is:

C:\Program Files\NCR\Teradata Client\tpump\tpumpar.csq

The monitor interface is implemented by providing a table (SysAdmin.TPumpStatusTbl) in the RDBMS. TPump reads commands from this table and places update status in it. This table is required to use the monitor functionality but is otherwise optional.

Checking Status of a Job

TPump users can find out the status of an import by querying against this table. TPump updates this table approximately once every minute.

Changing the Statement Rate as a Job Runs

TPump users can alter the statement rate of an import by updating this table. TPump examines this table approximately once every minute.



TPump Monitor

Tool to control and track TPump imports.

- The table `SysAdmin.TPumpStatusTbl` is updated once a minute.
- Alter the statement rate on an import by updating this table using macros.
- Use macros and views to access this table.

DBA Tools

View

- `SysAdmin.TPumpStatus` - view allows DBAs to view all of the TPump jobs.

Macro

- `SysAdmin.TPumpUpdateSelect` - allows DBAs to manage individual TPump jobs.

User Tools

View

- `SysAdmin.TPumpStatusX` - allows users to view their own TPump jobs.

Macro

- `TPumpMacro.UserUpdateSelect` - allows users to manage their own jobs.

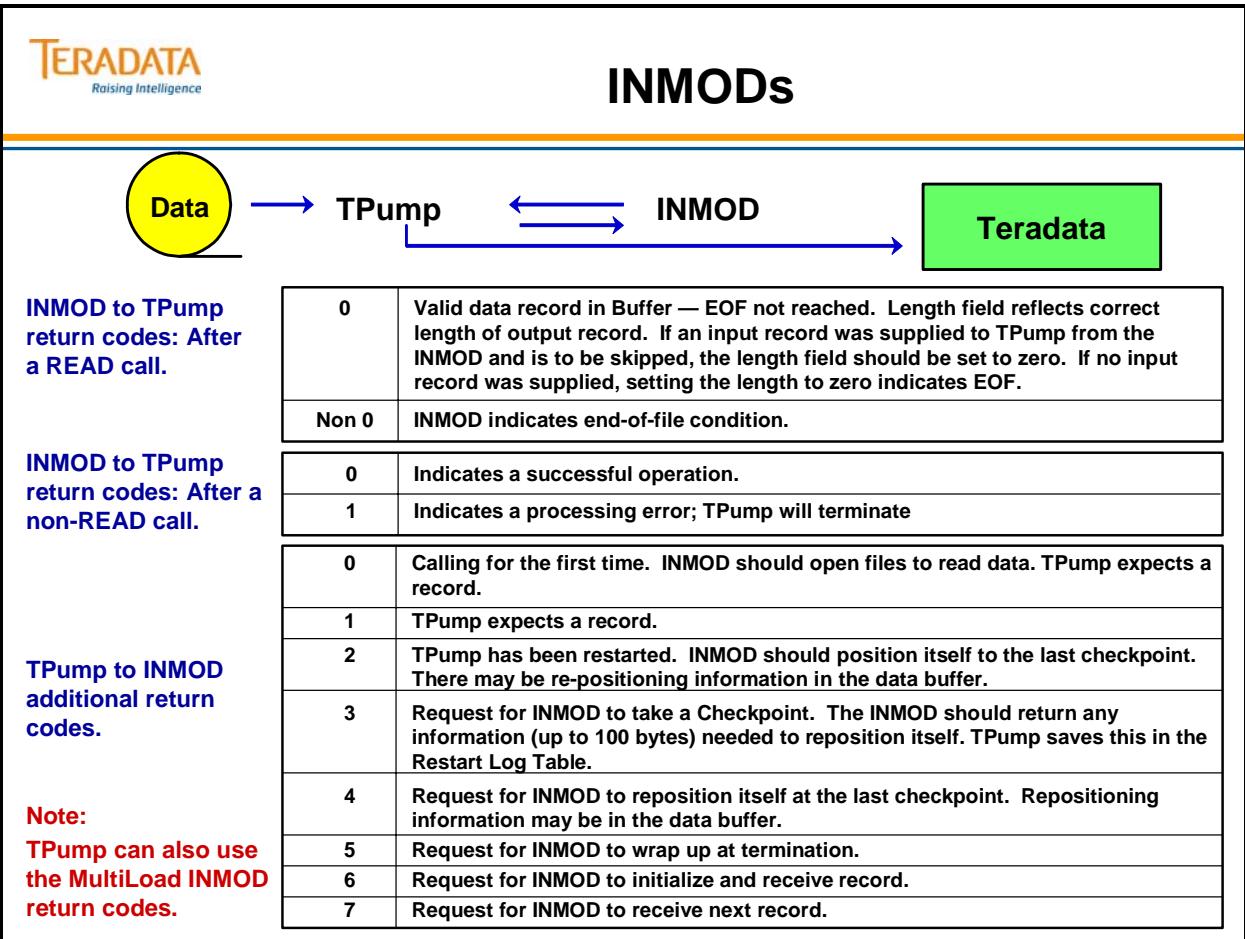
INMOD

An INMOD is a user exit routine used by TPump to supply or preprocess input records. The INMOD is specified as part of the IMPORT command.

Major functions performed by an INMOD include:

- Generating records to be passed to TPump.
- Validating a data record before passing it to TPump.
- Reading data directly from one or more database systems like IMS, Total.
- Converting fields in a data record before passing it to TPump.

Because of operational differences between TPump and the older utilities, some changes have been made to the INMOD utility interface for TPump. For compatibility with INMODs, the FDLINMOD parameter should be used. The use of this parameter provides support of existing INMODs, with the some restrictions, as noted in the TPump manual.



Application Utility Checklist

The facing page adds the TPump capabilities to the checklist.



Application Utility Checklist

Feature	BTEQ	FastLoad	FastExport	MultiLoad	TPump
DDL Functions	ALL	LIMITED	Yes (SE)	Yes (SE)	Yes (SE)
DML Functions	ALL	INSERT	SELECT	INS/UPD/DEL	INS/UPD/DEL
Multiple DML	Yes	No	Yes	Yes	Yes
Multiple Tables	Yes	No	Yes	Yes	Yes
Multiple Sessions	Yes	Yes	Yes	Yes	Yes
Protocol Used	SQL	FASTLOAD	EXPORT	MULTILOAD	SQL
Conditional Expressions	Yes	No	Yes	Yes	Yes
Arithmetic Calculations	Yes	No	Yes	Yes	No
Data Conversion	Yes	1 per column	Yes	Yes	Yes
Error Files	No	Yes	No	Yes	Yes
Error Limits	No	Yes	No	Yes	Yes
User-written Routines	No	Yes	Yes	Yes	Yes
Support Environment (SE)	No	No	Yes	Yes	Yes

Summary

The facing page summarizes some of the important concepts regarding the TPump utility.



Summary

- Allows near real-time updates from transactional systems into the warehouse.
- Performs INSERTs, UPDATEs, DELETEs, or UPSERTs.
- Alternative to MultiLoad for low-batch maintenance of large databases.
- Uses row-hash locks, allowing concurrent updates on the same table.
- Can always be stopped and locks dropped with no ill effect.
- User can specify how many updates occur minute by minute; can be changed as the job runs.
- No arithmetic functions or file concatenations.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Match the item in the first column to its corresponding statement in the second column.

- | | |
|---|---|
| <input type="checkbox"/> 1. TPump purpose | A. Query against TPump status table |
| <input type="checkbox"/> 2. MultiLoad purpose | B. Concurrent updates on same table |
| <input type="checkbox"/> 3. Row hash locking | C. Low-volume changes |
| <input type="checkbox"/> 4. PACK | D. Use to specify how many statements to put in a multi-statement request |
| <input type="checkbox"/> 5. MACRO | E. Large volume changes |
| <input type="checkbox"/> 6. Statement rate change | F. Used instead of DML |

Lab Exercise 39-1

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

```
.LOGON    ...;  
.EXPORT DATA FILE = data39_1;  
EXEC AP.Lab39_1;  
.EXPORT RESET  
.LOGOFF;
```

The size of data39_1 should be 15,600 bytes.

A technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function.

Switch to your terminal window where UNIX is running and ...

3. **cat > lab39_13.tpp** (or whatever filename you wish)

Use the mouse to choose the **Edit ➔ Paste** function

To exit the cat command, press either the DELETE key or CNTL C.



Lab Exercises

Lab Exercise 39-1

Purpose

In this lab, you will perform an operation similar to lab 38-2, using TPump instead of MultiLoad. For this exercise, use a PACK of 20 and a RATE of 4800.

What you need

Data file (*data39_1*) created from macro AP.Lab39_1.

Tasks

1. Delete all rows from the Accounts Table and use the following INSERT/SELECT to create 100 rows of test data:

```
INSERT INTO Accounts SELECT * FROM AP.Accounts WHERE Account_Number LT 20024101 ;
```

2. Export data to the file *data39_1* using the macro AP.lab39_1.
3. Prepare a TPump script which performs an UPSERT operation (INSERT MISSING UPDATE) on your Accounts table as a single operation. Use the data from *data39_1* as input to the UPSERT script. If the row exists, UPDATE the Balance_Current with the appropriate incoming value. If not, INSERT a row into the Accounts table. In your script, be sure to set a statement rate.
4. Run the script.
5. Validate your results. TPump should have performed 100 UPDATES and 100 INSERTS with a final return code of zero.

Teradata Training

Notes

Module 40



Choosing a Utility

After completing this module, you will be able to:

- **Describe various solutions to applications.**
- **Compare how different utilities work for the same application.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Maximum Number of Load Jobs	40-4
Maximum Number of Load Jobs (cont.).....	40-6
Solution 1: Update or Delete vs. Insert>Select	40-8
Solution 2: SQL Update vs. MultiLoad or TPump	40-10
Solution 3: SQL Update vs. FastLoad.....	40-12
Utility Considerations	40-14
Various Ways of Performing an Update	40-16
Choosing a Utility Exercise	40-18

Maximum Number of Load Jobs

With Teradata V2R6.0 (and previous releases), the DBS Control parameter MaxLoadTasks has a maximum limit of 15. One of the reasons that this limit is enforced is to prevent FastLoad, MultiLoad, and FastExport jobs from using up all available AMP worker tasks (AWTs).

In Teradata Database V2R6.1, the maximum number of concurrent load jobs is increased.

- For FastLoad and MultiLoad Jobs: up to 30 concurrent jobs
- For FastExport Jobs: Up to 60 jobs (minus the number of active FastLoad and MultiLoad jobs)

Users should be aware that running more load/unload utilities may impact other work and applications running concurrently (such as DSS queries or tactical queries) because of higher demand of the following resources: number of sessions, CPU, I/O, and memory.

This new feature is controlled by a new internal DBS Control parameter named MaxLoadAWT. (AWT – AMP Worker Tasks).

When MaxLoadAWT is zero (by default), this feature is disabled. Everything works as before. The DBS control flag MaxLoadTasks which specifies the concurrency limit for FastLoad, MultiLoad, and FastExport cannot be greater than 15.

When MaxLoadAWT is greater than zero, this feature is effectively enabled. This parameter specifies the maximum number of AWTs that can be used by FastLoad and MultiLoad jobs. The maximum allowable value is 60% of the total AWTs. Usually the maximum number of AWTs is 80; therefore, this maximum is 48.

Characteristics of this feature (when enabled) include:

- MaxLoadTasks only controls FastLoad and MultiLoad jobs. Its maximum limit is increased from 15 to 30.
- FastExport jobs are managed differently:
 - FastExport is no longer controlled by MaxLoadTasks flag.
 - FastExport limit is 60 minus number of active FastLoad and MultiLoad jobs.
 - A FastExport job is only rejected if the total number of active utility jobs is 60.
 - The minimum number of FastExport jobs that can run is 30. A FastExport job may be able to run even when FastLoad and MultiLoad jobs are rejected.



Maximum Number of Load Jobs

Teradata V2R6.0 (and previous releases)

- Maximum number of concurrent FastLoad, FastExport, and MultiLoad jobs is determined by a DBS Control parameter – **MaxLoadTasks** (range of 0 to 15).

Teradata V2R6.1 Feature

- A higher maximum number of concurrent FastLoad, FastExport, and MultiLoad jobs is possible and is determined by a new DBS Control parameter – **MaxLoadAWT**. This parameter specifies the maximum # of AMP Worker Tasks available to load utilities.
- If **MaxLoadAWT = 0** (the default), then ...
 - MaxLoadTasks has a range of 0 to 15.
 - MaxLoadTasks controls the maximum number of concurrent FastLoad, MultiLoad, and FastExport jobs.
 - Effectively works the same as previous releases.
- If **MaxLoadAWT > 0** (maximum value is 48), then ...
 - MaxLoadTasks has a range of 0 to 30.
 - Maximum number of total load jobs is 60.
 - MaxLoadTasks ONLY impacts FastLoad and MultiLoad jobs.
 - FastExport is no longer controlled by MaxLoadTasks flag.
 - FastExport limit is 60 minus number of active FastLoad and MultiLoad jobs.

Maximum Number of Load Jobs (cont.)

If the MaxLoadAWT parameter is greater than 0, then the maximum number of concurrent load utilities is controlled by both MaxLoadTasks and MaxLoadAWT parameters.

A new FastLoad or MultiLoad job is allowed to start only if BOTH MaxLoadTasks AND MaxLoadAWT limits are not reached.

FastLoad and MultiLoad jobs use a different number of AWTs depending on the phase the utility is in.

<u>Utility</u>		<u># of AWTs needed</u>
FastLoad	Phase 1 (Acquisition)	3
	Phase 2 (Sort)	1
MultiLoad	Acquisition Phase	2
	Application Phase	1
FastExport	All phases	0*

* The SELECT phase of FastExport actually uses 2 AWTs, but these are executed as normal DML so no AWT is counted toward the AWT limit. The export phase is processed by the Load Control Task (LCT) so no AWT is required.

For example, with MaxLoadAWT = 48 and MaxLoadTasks = 30, possible job mixes include:

- 16 FastLoad jobs in Phase 1, or
- 9 FastLoad jobs in Phase 1 and 21 FastLoad jobs in Phase 2, or
- 24 MultiLoad jobs in Acquisition Phase, or
- 5 MultiLoad jobs in Acquisition Phase and 25 MultiLoad jobs in Application Phase



Maximum Number of Load Jobs (cont.)

If **MaxLoadAWT > 0**, then the maximum number of concurrent load utilities is controlled by [both MaxLoadTasks and MaxLoadAWT](#).

- A new FastLoad or MultiLoad job is allowed to start only if BOTH MaxLoadTasks AND MaxLoadAWT limits are not reached.
 - Therefore, a job may be rejected before MaxLoadTasks limit is exceeded.
- A new FastExport job is only rejected if the total number of utility jobs is 60.

FastLoad and MultiLoad jobs use a different number of AWTs depending on the phase the utility is in.

<u>Utility</u>		<u># of AWTs needed</u>
FastLoad	Phase 1 (Acquisition)	3
	Phase 2 (Sort)	1
MultiLoad	Acquisition Phase	2
	Application Phase	1
FastExport		0

For example, with MaxLoadAWT = 48 and MaxLoadTasks = 30, you can only have:

- | | |
|---|-----------|
| 10 FastLoad jobs in Phase 1 | – 30 AWTs |
| 9 MultiLoad jobs in the Acquisition Phase | – 18 AWTs |

Solution 1: Update or Delete vs. Insert/Select

As fast or as flexible as the application utilities might be, you sometimes need to employ a certain amount of ingenuity and imagination to make the best use of them.

Consider the simple global update on the facing page. It is a long-running table update that requires maintenance of a transient journal row for every row changed. In the event of failure, it will need to roll back the transaction as if it had never begun. The longer the transaction takes to complete, the greater the chance of failure.

Another way of performing this task might be to create a new table and populate it using the "fast path" INSERT/SELECT.

Another approach you might consider is to delete unneeded rows from a table and preserve the rows you wish to keep.

By looking at the problem in a different way, you can use the fast INSERT/SELECT performance to handle a series of updates.



Solution 1: Update or Delete vs. Insert/Select

Update all customers credit limit by 20%.

```
UPDATE Customer
SET Credit_Limit = Credit_Limit * 1.20 ;
```

Delete all transactions prior to 2000.

```
DELETE FROM Trans
WHERE Trans_Date < DATE '2000-01-01';
```

- SQL statements are treated as single transactions.
- This requires Transient journal space for every updated or deleted row in the target table until the transaction finishes.**
- A failure will cause the system to back out all changes that have been completed (which could take hours if the table and the number of completions are very large).

```
CREATE TABLE Cust_N AS Customer WITH NO DATA ;
INSERT INTO Cust_N
    SELECT ... , Credit_Limit * 1.20 FROM Customer ;
DROP TABLE Customer ;
RENAME TABLE Cust_N TO Customer ;
```

```
CREATE TABLE Trans_N AS Trans WITH NO DATA;
INSERT INTO Trans_N SELECT * FROM Trans
    WHERE Trans_Date > DATE '1999-12-31';
DROP TABLE Trans;
RENAME TABLE Trans_N TO Trans;
```

- This approach reduces the number of changes that would have to be backed out by the system in case of a failure.
- The “Fast path” INSERT / SELECT offers the fastest possible transfer of data that can be achieved in a single SQL statement.
- The same approach could also be used to delete rows from a table, simply by not selecting them for insert.

Solution 2: SQL Update vs. MultiLoad or TPump

The facing page displays another solution that uses the speed and sophisticated functionality of MultiLoad to attain good performance by writing updates to disk a block at a time, effectively removing the disk as a performance constraint.

If the percentage of updates is small as compared to the number of rows in the table, TPump may be a good choice.



Solution 2: SQL Update vs. MultiLoad or TPump

SQL Update

```
UPDATE Customer
  SET credit_limit      = credit_limit * 1.20
 WHERE over_limit_count > 0
   AND late_payment_count = 0 ;
```

Full table Write lock

Full table scan

Potentially large Transient Journal

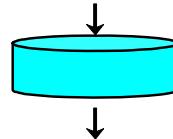
FastExport

```
SELECT customer_number, phone, credit_limit * 1.20
  FROM Customer
 WHERE over_limit_count > 0
   AND late_payment_count = 0 ;
```

Full table Read lock

Full table scan

No Transient Journal



MultiLoad or TPump

```
UPDATE Customer
  SET credit_limit      = :credit_limit
 WHERE phone            = :phone
   AND customer_number   = :customer_number;
```

MultiLoad advantages -

- Sorts updates by Primary Index.
- Each data block is accessed only once.
- Full automatic restart under all conditions.

If the percentage of updates compared to number of table rows is large, use MultiLoad. If the percentage is small, use TPump.

Solution 3: SQL Update vs. FastLoad

FastLoad is limited in functionality in that, like the optimized INSERT/SELECT, it is used to insert rows into an initially empty table.

Even so, FastLoad is very fast. If you export the rows you wish to update to the host (updating the values in the process), and add the remainder of the rows, you have a complete host-resident file suitable for recovery. You can then take full advantage of FastLoad's excellent performance to insert all the rows into a new table:

- Create a new table.
- Use FastLoad to populate it at high speed.
- Drop the old table.
- Rename the new table to the old table name.

Again, by looking at the problem from a different perspective, you are able to employ the high speed of FastLoad to perform updates to a populated table.

Note: The examples shown for all solutions illustrate *updates* to the target table. Each of the solutions could equally be changed to *delete* rows from the target table.



Solution 3: SQL Update vs. FastLoad

```
SQL Update
UPDATE Customer
  SET credit_limit      = credit_limit * 1.20
 WHERE over_limit_count > 0
   AND arrears_count     = 0 ;
```

Full table Write lock

Full table scan

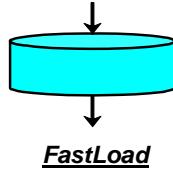
Potentially large Transient Journal

```
FastExport
SELECT . . . , credit_limit * 1.20, . . .
  FROM Customer
 WHERE over_limit_count > 0
   AND arrears_count     = 0
; SELECT *
   FROM Customer
 WHERE NOT (over_limit_count > 0 AND
            arrears_count = 0);
DELETE FROM Customer;
```

Full table Read lock

Full table scan

No Transient Journal



An alternate solution when the external disk space is capable of housing the entire table.

Utility Considerations

While a selection of tools is available for batch-type processing, application success relies principally on choosing the right one.

In order to make a good choice, you have to ask the right questions. Some of these are obvious, such as “Is the utility supported on the host ?” Other important concerns are less apparent: “What happens when things go wrong?” “Does this utility allow me to work within the window required by the user?”

To get a count of load jobs that are currently executing, you can use the following SQL.

```
SELECT      COUNT(DISTINCT LogonSequenceNo) AS Utility_Cnt  
FROM        DBC.SessionInfo  
WHERE       Partition IN ('Fastload', 'Export', 'MLoad');
```



Utility Considerations

- **Utility support**
 - Has the customer purchased the utility and does it run on your host?
- **Restart capability**
 - Is there a restart log?
 - What happens with a Teradata restart?
 - What happens if the host fails?
- **Multiple sessions**
 - Does the utility support multiple sessions?
 - How do you choose the optimum number?
- **Error handling**
 - Are errors captured in an error file?
 - Do you have control over error handling?
- **Are special processing routines needed?**
 - Does the utility support INMODs, OUTMODs, or AXSMODs?
- **Does the utility meet performance requirements?**
 - Does the job fit your batch window?
 - Do the tables require continuous (7 x 24) access by the user groups?

Various Ways of Performing an Update

Consider this simple global update of a large table. We use a bank application since you are probably familiar with bank accounts, checks, deposits, etc.

At the end of each month, after it prints the bank statements, the bank (which maintains derived data for the account balance) is required to set the value of the Balance_Forward to Balance_Current. The data is already available in the Teradata database.

Based on what you have learned, try to evaluate the listed methods fastest to slowest.



Various Ways of Performing an Update

Prior to producing a monthly statement, the Bank sets the Balance_Forward amount equal to the Balance_Current for 900,000 accounts:

```
UPDATE Accounts  
SET Balance_Forward = Balance_Current ;
```

Which is the fastest method?

1. Submit the statement above.
2. INSERT/SELECT revised values to a new table, drop the original table and rename the new table.
3. EXPORT data to the host and UPDATE using MultiLoad.
4. EXPORT the required values to the host and FastLoad them back to a new table. Drop the old table and rename the new.

Order the above Fastest to Slowest _____ , _____ , _____

Choosing a Utility Exercise

The facing page contains two scenarios. Make the best choice of which utilities to use for each of the two scenarios.



Choosing a Utility Exercise

1. A sales table currently contains 24 months of transaction data. At the end of each month, 25 million rows are added for the current month and 25 million rows are removed for the oldest month. There is enough PERM space to hold 30 months worth of data.

Which choice (from below) makes the most sense? _____

2. The customer decides to partition the table by month and maintain each month's data in a separate partition. At this time, only the most recent 24 months need to be maintained. At the end of each month, data is loaded into a new monthly partition and the oldest month is removed.

Which choice (from below) makes the most sense? _____

Utility Choices:

- A. Use FastLoad to add new data to existing table, and ALTER TABLE to remove old data.
- B. Use MultiLoad to add new data to existing table, and ALTER TABLE to remove old data.
- C. Use FastLoad to add new data to existing table, and MultiLoad to remove old data.
- D. Use MultiLoad to add new data to existing table, and MultiLoad to remove old data.
- E. Use TPump to add new data, and TPump to remove old data.
- F. Use TPump to add new data, and ALTER TABLE to remove old data.

Teradata Training

Notes

Module 41



Database Administration and Building the Database Environment

After completing this module, you should be able to:

- Describe the purpose and function of an administrative user.
- Differentiate between creators, owners (parents), and children.
- Describe how to transfer ownership of databases and users.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Database Administration.....	41-4
Initial Teradata Database.....	41-6
Administrative User	41-8
Owners, Parents and Children.....	41-10
Creating New Users and Databases	41-12
Transfer of Ownership	41-14
DELETE/DROP Statements	41-16
Teradata Administrator – New System	41-18
Teradata Administrator – Hierarchy	41-20
Summary	41-22
Review Questions	41-24

Database Administration

The facing page identifies some of the functions of a Teradata Database Administrator (DBA). These functions include:

- User Management – creation of databases, users, accounts, roles, and profiles
- Space Allocation and Usage – perm, spool, and temporary space
- Access of Objects (e.g., tables, views) – access rights, roles, use of views, etc.
- Access Control and Security – logon access, logging access, etc.
- System Maintenance – specification of system defaults, restarts, data integrity, etc.
- System Performance – use of Priority scheduler, job scheduling (TDQM), etc.
- Resource Monitoring – use of ResUsage tables/views, query capture (DBQL), etc.
- Data Archives, Restores, and Recovery – ARC facility, Permanent Journals, etc.

Examples of tools available to the Teradata DBA include:

- Use of Data Dictionary/Directory tables and views to manage the system
- Teradata Administrator – graphical Windows tool to assist in administration
- Teradata Manager – suite of Windows tools for performance management, etc.
- Teradata Analyst Toolset – Visual Explain, Index Wizard, Statistics Wizard, and Teradata SET
- Teradata Dynamic Workload Manager, Teradata Workload Analyzer, and Query Scheduler – job scheduling facility
- System utilities – e.g., dbscontrol, ferret, rebuild, etc.
- User scripts and 3rd party applications

To do these functions, it is important to understand key concepts such as the Teradata hierarchy and the concepts of ownership (parents and children). The hierarchy and the concept of ownership will be discussed in this module.

Acronyms:

- Teradata DWM – Teradata Dynamic Workload Manager
DBQL – Database Query Log
DBW – Database Window
SET – System Emulation Tool



Database Administration

Some of the functions of a Teradata Database Administrator (DBA) include:

- User and Database Management
- Space Allocation and Usage
- Access of Objects (e.g., tables, views, macros, etc.)
- Access Control and Security
- System Maintenance
- System Performance
- Resource Monitoring
- Data Archives, Restores, and Recovery

Examples of tools available to the Teradata DBA include:

- Use of Data Dictionary/Directory tables and views to manage the system
- Teradata Administrator – Windows administration utility
- Teradata Manager – suite of Windows tools – e.g., Teradata Performance Monitor
- Teradata Analyst Toolset – Index Wizard, Statistics Wizard, and Teradata SET
- Teradata Dynamic Workload Manager and Teradata Workload Analyzer
- Database Console Window and System utilities – e.g., dbscontrol, ferret, rebuild, etc.
- User scripts and 3rd party applications

To do these functions, it is important to understand key concepts such as ...

- Teradata hierarchy and the concepts of ownership (parents and children)

Initial Teradata Database

The Teradata Database software includes the following users and databases:

DBC

With the few exceptions described below, a system user named DBC owns all usable disk space. DBC's space includes dictionary tables, views and macros discussed in the next module. The usable disk space of DBC initially reflects the entire system hardware capacity, less the following:

Sys_Calendar

This user is used to hold the system calendar table and views. The initial MAXPERM value is 15,000,000 bytes.

SysAdmin

SysAdmin is a system user with a minimum of space for table storage. The initial MAXPERM value is 40,000,000 bytes. SYSADMIN contains several supplied views and macros as well as a restart table for network-based FastLoad jobs.

SystemFE User

SystemFE is a system user delivered with a small amount of MAXPERM space (60,000,000 bytes) for tables. It contains special macros used to generate diagnostic reports for Customer Engineers (field support personnel) logged on as this user. The default password is "service". Following a SYSINIT, approximately 100 KB of the 60 MB is used.

Crashdumps User

Crashdumps is a user provided for temporary storage of PDE dumps generated by the software. The default is 1 GB. Crashdumps is created within system user DBC and its space is allocated from current PERM space for DBC. You should enlarge the Crashdumps user based on the size of the configuration to accommodate at least three dumps.

PUBLIC and TDPUSER

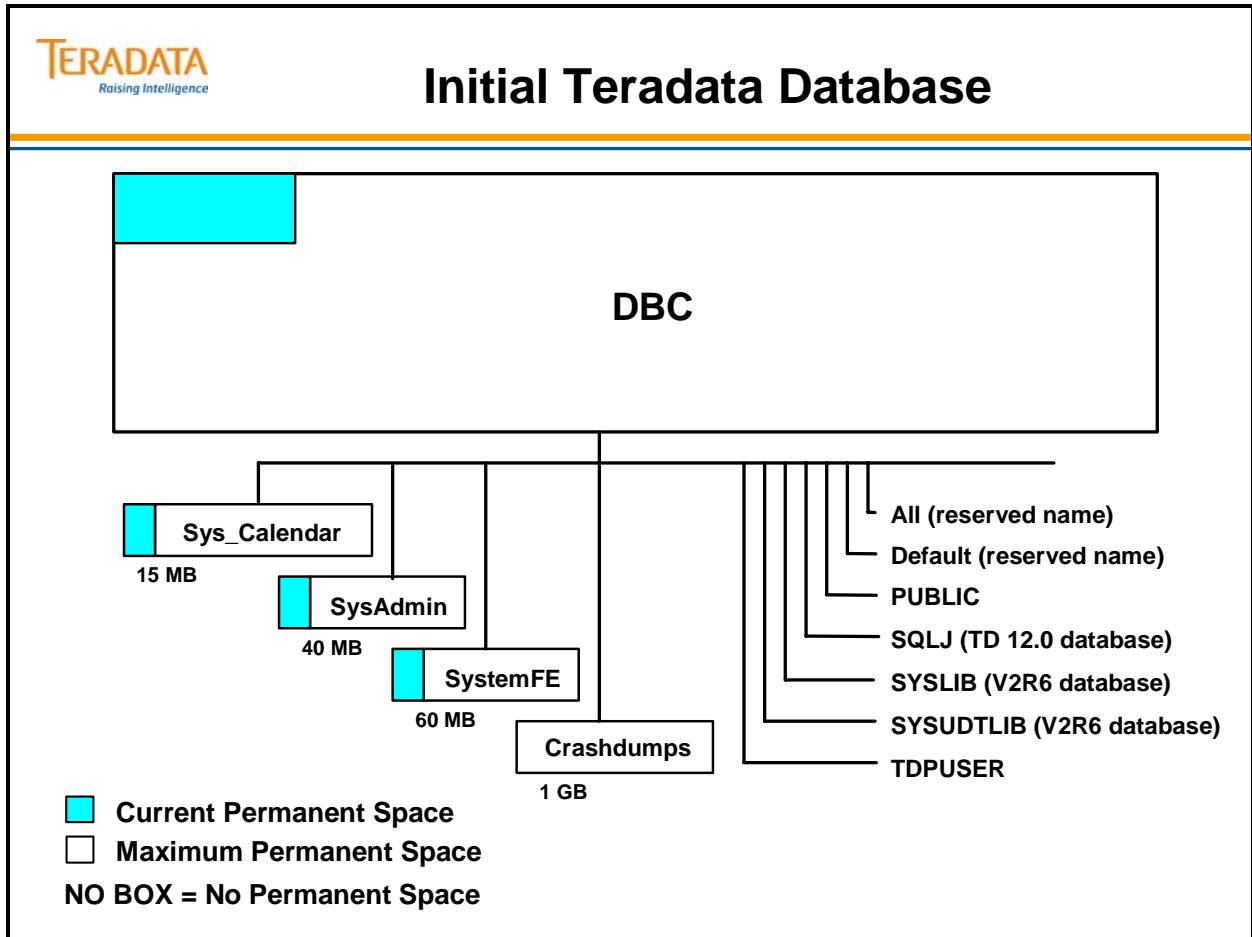
PUBLIC and TDPUSER are "dummy" database names used by the database system software and appear in the system hierarchy. These users are defined with no permanent space. TDPUSER is used to support two-phase commit.

Default and All

Default and All are also "dummy" database names that are reserved by the database system software and don't appear in the hierarchy.

SQLJ, SYSLIB, and SYSUDTLIB Databases

The SYSLIB database can be used to store user-defined functions and the SYSUDTLIB database can be used to store user-defined data types. The SQLJ database (new with Teradata 12.0) database contains a series of new views that reference the new dictionary tables to support Java external stored procedures.



Administrative User

System user DBC contains all Teradata Database software components and all system tables.

Before you define application users and databases, you should first use the CREATE USER statement to create a special administrative user to complete these tasks.

The amount of space for the administrative user is allocated from DBC's current PERM space. DBC becomes the owner of your administrative user and of all users and databases you subsequently create.

Be sure to leave enough space in DBC to accommodate the growth of system tables and logs, and the transient journal.

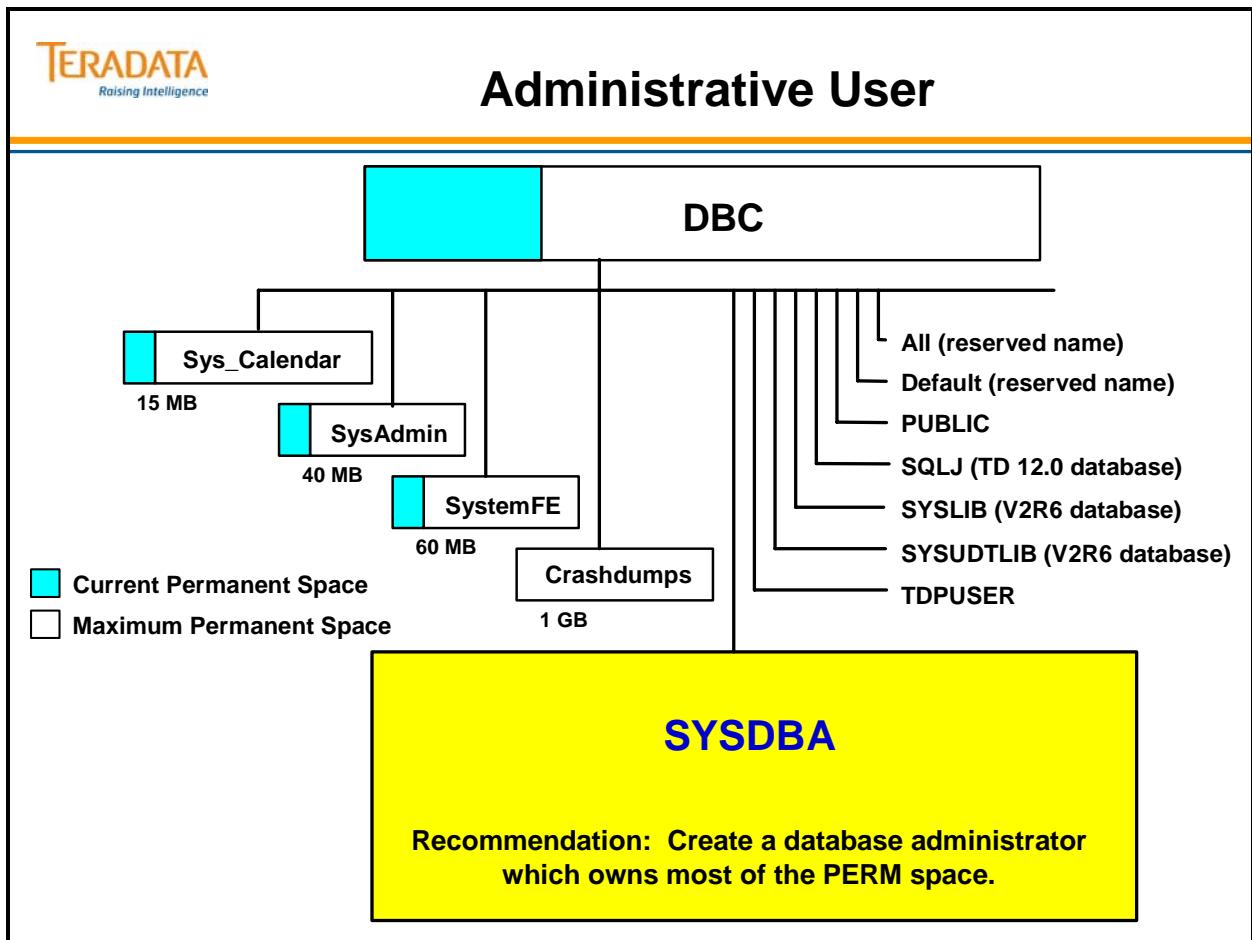
(You can name the user anything you would like. We have called the user SYSDBA.)

Create the administrative user, and then logon as that user to protect sensitive data in DBC. In addition, change and secure the DBC password.

To ensure perm space is from the administrative user, logon as that user to add other users and databases.

Notes:

- All space in the Teradata Database is owned. No disk space known to the system is unassigned or not owned.
- Think of a user as a database with a password. Both may contain (or "own") tables, views and macros.
- Both users and databases may hold privileges.
- Only users may logon, establish a session with the Teradata Database, and submit requests.



Owners, Parents and Children

As you define users and databases, a hierarchical relationship among them will evolve.

When you create new objects, you subtract permanent space from the assigned limit of an existing database or user. A database or user that subtracts space from its own permanent space to create a new object becomes the immediate owner of that new object.

An “owner” or “parent” is any object above you in the hierarchy through which a direct line of ownership is established during the creation process. (Note that you can use the terms owner and parent interchangeably.) A “child” is any object below you in the hierarchy through which a direct line of ownership is established during the creation process.

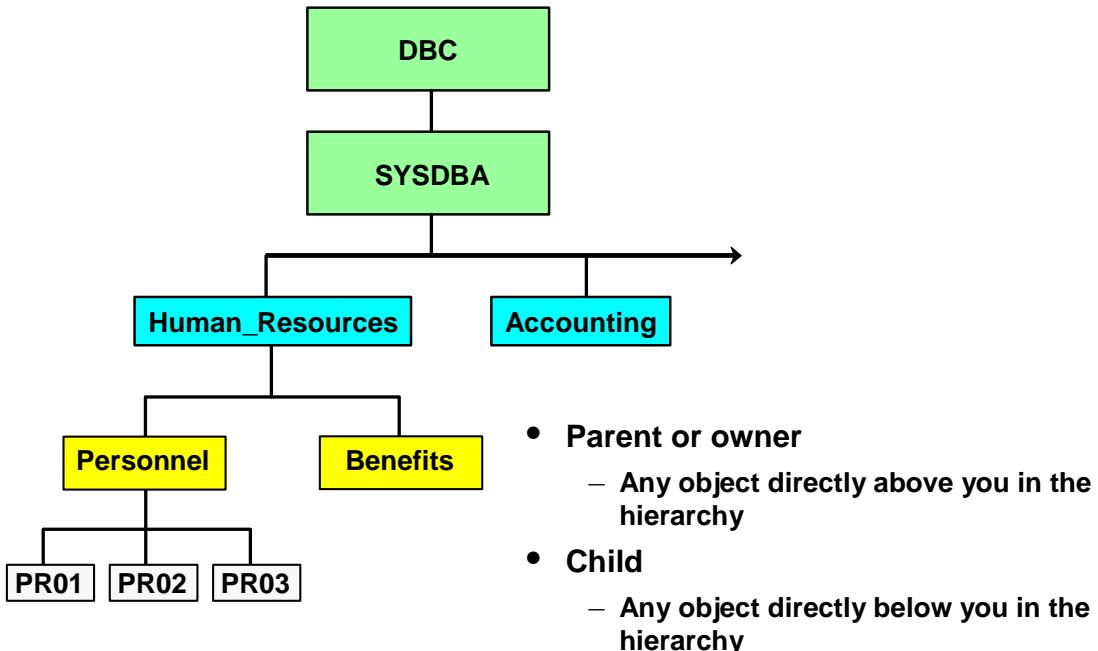
The term “immediate parent” is sometimes used to describe a database or user just above you in the hierarchy.

Example

The diagram on the facing page illustrates Teradata system hierarchy. System user DBC is the owner, or parent, of all the objects in the hierarchy. The administrative user (SYSDBA) is the owner of all objects below it, such as Human Resources, Accounting, Personnel and Benefits. These objects are also children of DBC, since DBC owns SYSDBA.



Owners, Parents, and Children



Creating New Users and Databases

The “creator” of an object is the user who submitted the CREATE statement.

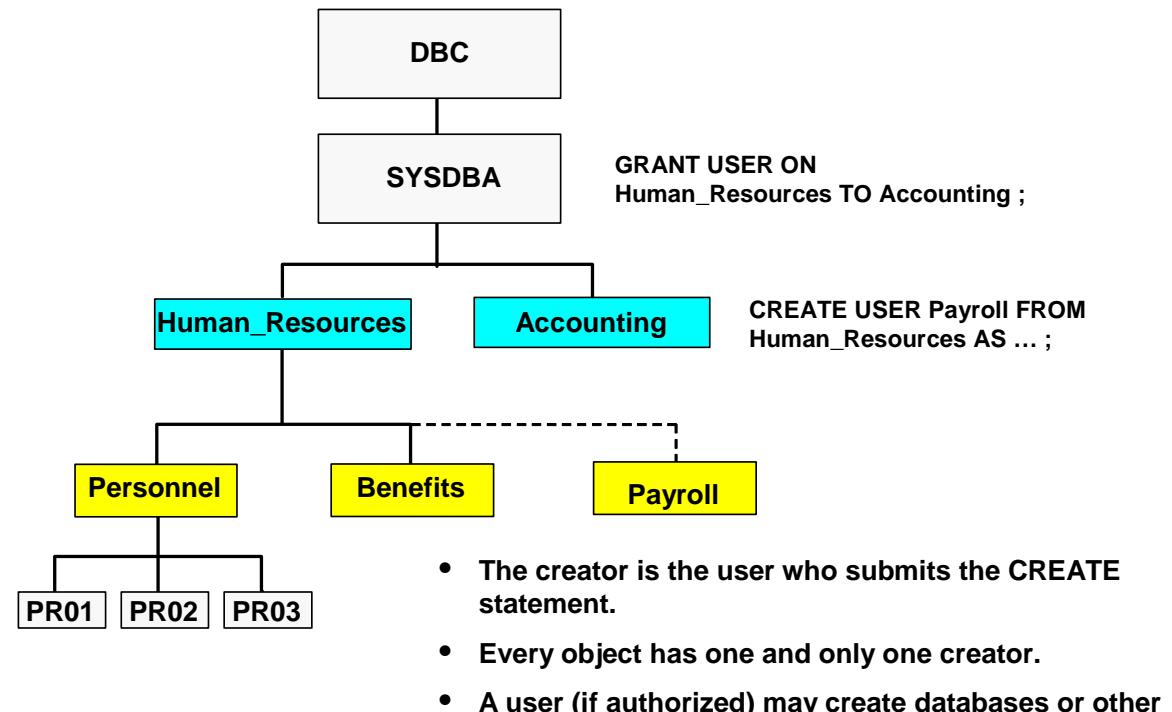
Every object has one and only one creator. If you are the creator of a new space, you automatically have access rights to that space and anything created in it.

Notes:

- While you may be the creator of an object, you are not necessarily the owner of the database or user that contains the object.
- You are the owner of an object if the new object is directly below you in the hierarchy.
- As a creator, you can submit a CREATE statement that adds a new object somewhere else in the hierarchy, assuming you have the appropriate privileges. In this instance, the creator (you) and the owner are two different users or databases.
- If authorized, you may create databases or users FROM someone else's space.
- You can transfer databases and users from one owner to another.



Creating New Users and Databases



Transfer of Ownership

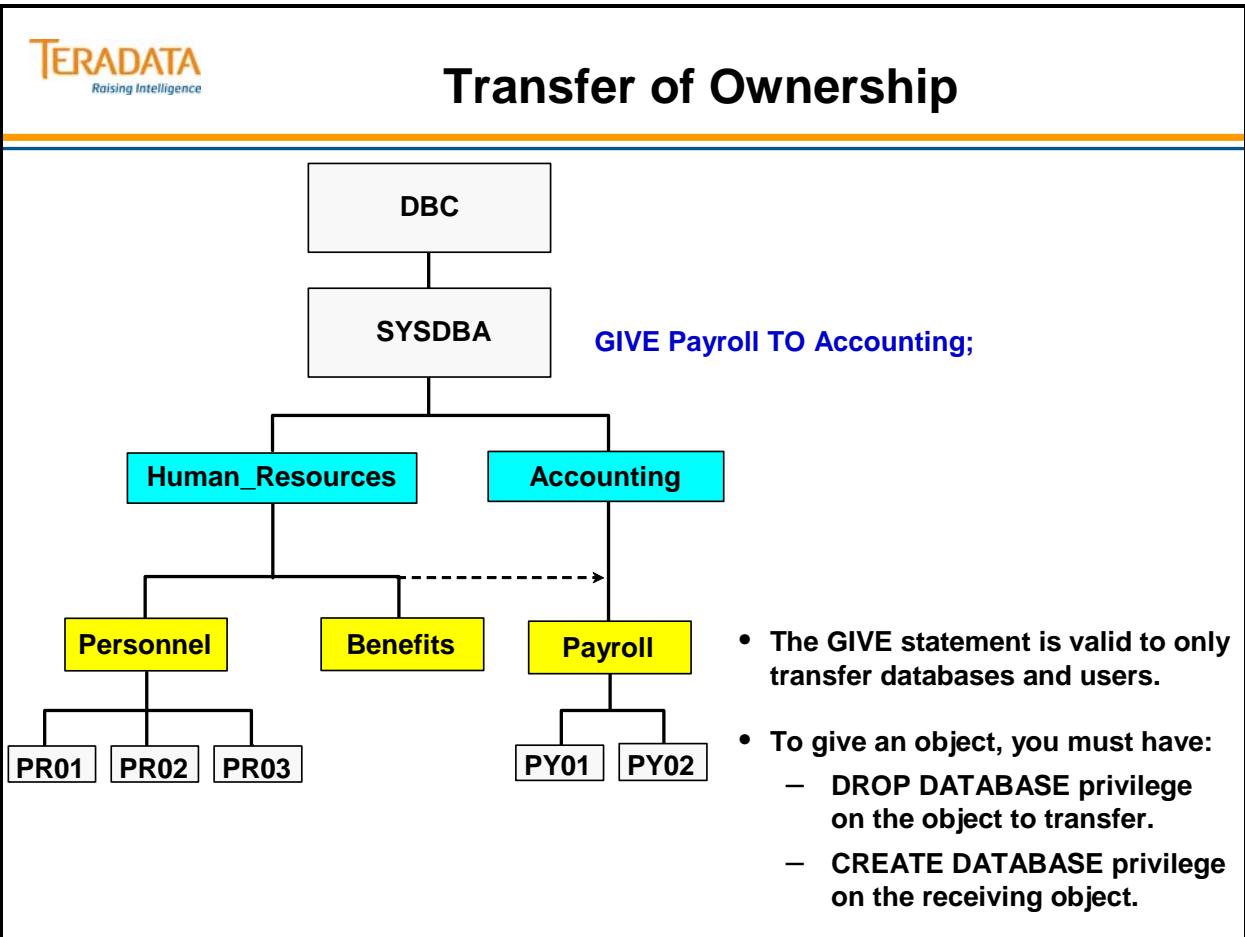
The GIVE statement transfers a database or user space to a recipient you specify. The GIVE statement also transfers all child databases and users as well as the tables, views and macros owned by the transferred object.

Rules affecting transfer of ownership:

- Use the GIVE statement to transfer databases and users only. (You cannot use the GIVE statement to transfer tables, views, and macros from one database to another.)
- To transfer an object, you must have DROP DATABASE privilege on the object to transfer and CREATE DATABASE privilege on the receiving object.
- You cannot give an object to one of its children.
- During a transfer, you transfer all objects the object owns.
- Transfer of ownership affects space ownership and access right privileges. When you transfer an object, the space the object owns is also transferred. The implications of how access rights are affected will be described in more detail later in this course.

Example

In the diagram on the facing page, the administrative user, SYSDBA, GIVES the user, Payroll, to Accounting. The original owner, Human_Resources, loses ownership of the perm space that belonged to Payroll. The new owner, Accounting, acquires ownership of the perm space that Payroll brings with it. All objects that belong to Payroll are transferred as well, including its tables, views, macros and children databases.



DELETE/DROP Statements

DELETE DATABASE and DELETE USER statements delete all data tables, views, and macros from a database or user. The database or user remains in the Teradata Database as a named object and retains the available space. None of that space is any longer in use. All space used by the deleted objects becomes available as spool space until it is reused as perm space.

You must have DROP DATABASE or DROP USER privilege on the referenced database or user to delete objects from them. The database or user that you are dropping cannot own other databases or users.

DELETE USER Example

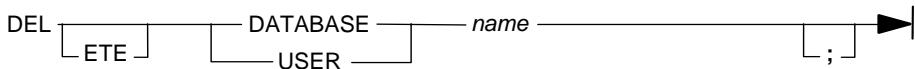
The diagram on the facing page illustrates a DELETE USER statement. Assume the user Personnel has three tables: TB01, TB02, and TB03. Human Resources logs on to the system and submits the DELETE USER statement on user Personnel. All tables are deleted from the user space owned by Personnel. The DELETE DATABASE/USER command does NOT delete a permanent journal, join indexes, or hash indexes.

DROP USER Example

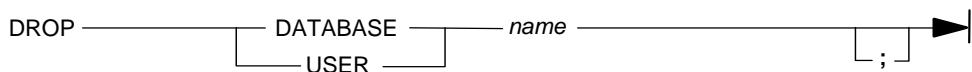
The DROP DATABASE or DROP USER statement drops empty databases or users only. You must delete all objects associated with the database or user before you can drop the DATABASE or USER. When you drop a database or user, its perm space is credited to the immediate owner.

The diagram on the facing page illustrates the DROP USER statement. Human Resources submits the DROP USER statement on user Personnel. The user Personnel is dropped from the hierarchy. The user space that belonged to user Personnel is returned to its parent, Human Resources.

DELETE USER Syntax

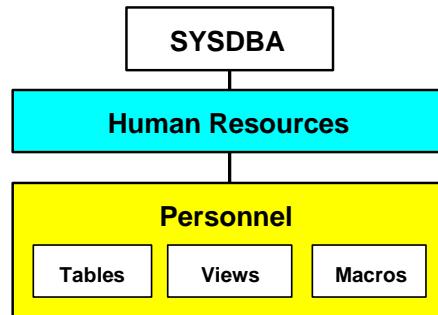


DROP USER Syntax

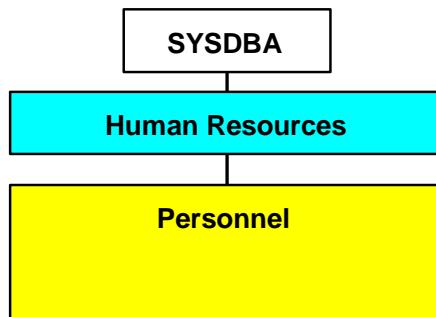




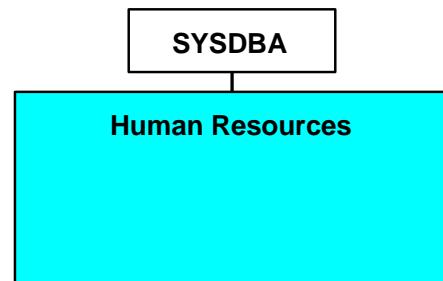
DELETE / DROP Statements



DELETE USER Personnel;



DROP USER Personnel;



Teradata Administrator – New System

Teradata Administrator (previously named WinDDI for Windows Data Dictionary Interface) is the Teradata Manager application that you can use to perform database administration tasks on the associated Teradata Database computer.

The facing page contains an example of the users and databases that exist in a newly initialized Teradata 12.0 system.

TERADATA
Raising Intelligence

Teradata Administrator New System

Teradata Administrator

- GUI interface to Teradata hierarchy and objects.
- This example shows the default users and databases in a newly initialized Teradata 12.0 system.

Which database has the majority of the Perm space?

The screenshot shows the Teradata Administrator interface with two main windows:

- DBC Table:** Shows a list of objects in the Training.DBC database. The columns are Name, Type, AccessCount, Last Access, Queue, Fallback, and Version. The data includes various macros like AccLogRule, ARC_NonEmpty_List, etc.
- Child Space - DBC Table:** Shows the current usage of space for different databases. The columns are Name, CurrentPerm, MaxPerm, PeakPerm, MaxSpool, and PeakSpool. The data shows that the DBC database is the largest consumer of permanent space.

Name	CurrentPerm	MaxPerm	PeakPerm	MaxSpool	PeakSpool
All	0	0	0	0	0
Crashdumps	0	1,024,000,000	0	1,024,000,000	0
DBC	21,121,536	1,333,761,255,206	120,095,232	1,335,510,255,206	27,554,304
Default	0	0	0	0	0
PUBLIC	0	0	0	0	0
SQLJ	0	600,000,000	0	1,335,510,255,200	0
SysAdmin	1,488,896	40,000,000	1,488,896	25,000,000	0
SYSLIB	6,144	10,000,000	6,144	1,335,510,255,200	0
SystemFe	149,504	60,000,000	149,504	1,335,510,255,200	0
SYSUDTLIB	0	0	0	1,335,510,255,200	0
Sys_Calendar	2,648,064	15,000,000	2,663,424	15,000,000	0
TDPUSER	0	0	0	1,335,510,255,200	0

479 rows returned Elapsed 00:00:01

Teradata Administrator – Hierarchy

The facing page contains an example of a screen display from a Teradata system and illustrates the hierarchy in the left pane.

You may use Teradata Administrator to perform the following functions:

- Create, modify and drop users or databases
- Create tables (using ANSI or Teradata syntax)
- Grant or revoke access/monitor rights
- Copy table, view or macro definitions to another database, or to another system
- Drop or rename tables, views or macros
- Move space from one database to another
- Run an SQL query
- Display information about a Database (list of tables, views, macros, child databases, rights)
- Display information about a Table (columns, journals, indexes, row counts, users, space summary), View (columns, info, rights, row count, users, show), or Macro (rights, users, info, show)

Teradata Administrator keeps a record of all the actions you take and can optionally save this record to a file. This record contains a time stamp together with the SQL that was executed, and other information such as the statement's success or failure.

To use the viewing functions of Teradata Administrator, you must have Select access to the DBC views of the Teradata DBS. To use the Copy, Drop, Create or Grant tools you must have the corresponding privilege on the table or database that you are trying to modify or create. To use the Browse or Row Count features you must have select access to the Table or View.

Additional examples of Teradata Administrator displays and capabilities will be shown in various modules throughout this course.

TERADATA
Raising Intelligence

Teradata Administrator – Hierarchy

Teradata 12.0 Administrator

- This example shows the hierarchy of a Teradata system and the objects in one of the databases.
- This utility also provides drag and drop capabilities.

The screenshot shows the Teradata Administrator interface. The left pane displays a tree view of the database hierarchy under the 'Training.DBC' database. The root node is 'DBC'. Other nodes include 'All', 'console', 'Crashdumps', 'DBCMANAGER', 'dbcmngr', 'Default', 'PUBLIC', 'Spool_Reserve', 'SQLJ', 'SysAdmin', 'Sysdba', 'Students', and 'Teradata_Training'. Under 'Teradata_Training', there is a node 'AP' which has children 'DS', 'PD', 'TFACT', 'tfact01', 'tfact02', 'tfact03', 'tfact04', 'tfact05', and 'tfact06'. The right pane is a grid table titled 'Teradata Administrator - [Training.DBC]' with columns: Name, Type, AccessCount, Last Access, Queue, Fallback, Version, and Te. It lists 13 rows of objects. A yellow callout box points to the left pane with the text: 'The hierarchy of the Teradata database is shown in the left pane.' A green callout box points to the 'TFACT' node in the tree with the question: 'Who is the immediate parent of TFACT?'.

Name	Type	AccessCount	Last Access	Queue	Fallback	Version	Te
1 LAB33_1	Macro			N	F	1	Te
2 LAB33_2	Macro			N	F	1	Te
3 LAB34_1_1	Macro			N	F	1	Te
4 LAB34_1_2	Macro			N	F	1	Te
5 LAB34_2	Macro			N	F	1	Te
6 LAB37_1	Macro			N	F	1	Te
7 LAB37_2	Macro			N	F	1	Te
8 LAB38_1	Macro			N	F	1	Te
9 LAB38_2	Macro			N	F	1	Te
10 LAB39_1	Macro			N	F	1	Te
11 Accounts	Table	10,000	2007-11-18 04:32:05	N	F	1	Te
12 Customer	Table	7,000	2007-11-18 04:32:57	N	F	1	Te
13 Trans	Table	15,000	2007-11-18 04:35:06	N	F	1	Te

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Initially, system user DBC owns all space in the Teradata Database except that owned by system users and databases.
- The database administrator should create a special administrative user containing most of the space available which will become the owner of all administrator-defined application databases and users.
- **Everyone directly higher in the hierarchy is a *parent or owner*. Everyone directly lower in the hierarchy is a *child*.**
- Every object has one and only one creator. The creator is the user who executes the CREATE statement.
- The GIVE statement enables you to transfer a database or user. The following privileges are necessary:
 - DROP DATABASE on the given object.
 - CREATE DATABASE on the receiving object.
- You cannot DROP databases or users that own objects (tables, views, macros, journals or children databases/users).
- Teradata Administrator provides an easy-to-use Windows-based graphical interface to the Teradata Database Data Dictionary.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. True or False. You should use system user DBC to create application users and databases.
2. True or False. An database or user can have multiple Owners, but only one Creator.
3. True or False. An Owner and a Parent are two different terms that mean the same thing.
4. True or False. An Owner and a Creator are two different terms that mean the same thing.
5. True or False. An administrative user (e.g., Sysdba) will never have more permanent space than DBC.
6. True or False. The GIVE statement transfers a database or user space to a recipient you specify. It does not automatically transfer all child databases of the transferred database or user.

Teradata Training

Notes

Module 42



The Data Dictionary

After completing this module, you will be able to:

- Summarize information contained in the Data Dictionary tables.
- Differentiate between restricted and unrestricted views.
- Use the supplied Data Dictionary views to retrieve information about created objects.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Data Dictionary / Directory.....	42-4
Fallback Protected Data Dictionary Tables.....	42-6
Non-Hashed Data Dictionary Tables	42-8
Updating Data Dictionary Tables	42-10
Supplied Data Dictionary Views.....	42-12
Restricted Views	42-14
Suffix Options with Views.....	42-16
Selecting Information about Created Objects	42-18
Children View	42-20
Databases View.....	42-22
Users View	42-24
Tables View	42-26
Columns View.....	42-28
Indices View.....	42-30
Indices View (Second Example).....	42-32
IndexConstraints View.....	42-34
ShowTblChecks View.....	42-36
ShowColChecks View	42-38
Triggers View.....	42-40
AllTempTables View	42-42
Referential Integrity Views.....	42-44
Using the DBC.Tables View	42-46
Referential Integrity States.....	42-48
DBC.All_RI_Children View.....	42-50
DBC.Databases2 View.....	42-52
Time Stamps in Data Dictionary.....	42-54
Teradata Administrator – List Columns of a View.....	42-56
Teradata Administrator – Object Options	42-58
Summary	42-60
Review Questions	42-62
Lab Exercise 42-1	42-64
Lab Exercise 42-2	42-70

Data Dictionary / Directory

The data dictionary/directory is a complete database composed of system tables, views, and macros that reside in system user DBC.

It is referred to as a Dictionary / Directory because it provides two functions:

- Dictionary – information you can view (e.g., you can view the columns and their attributes of a table)
- Directory – information to control the system (e.g., table names are converted to table IDs for the software to use)

The Teradata Data Dictionary / Directory is usually referred to as the Teradata Data Dictionary.

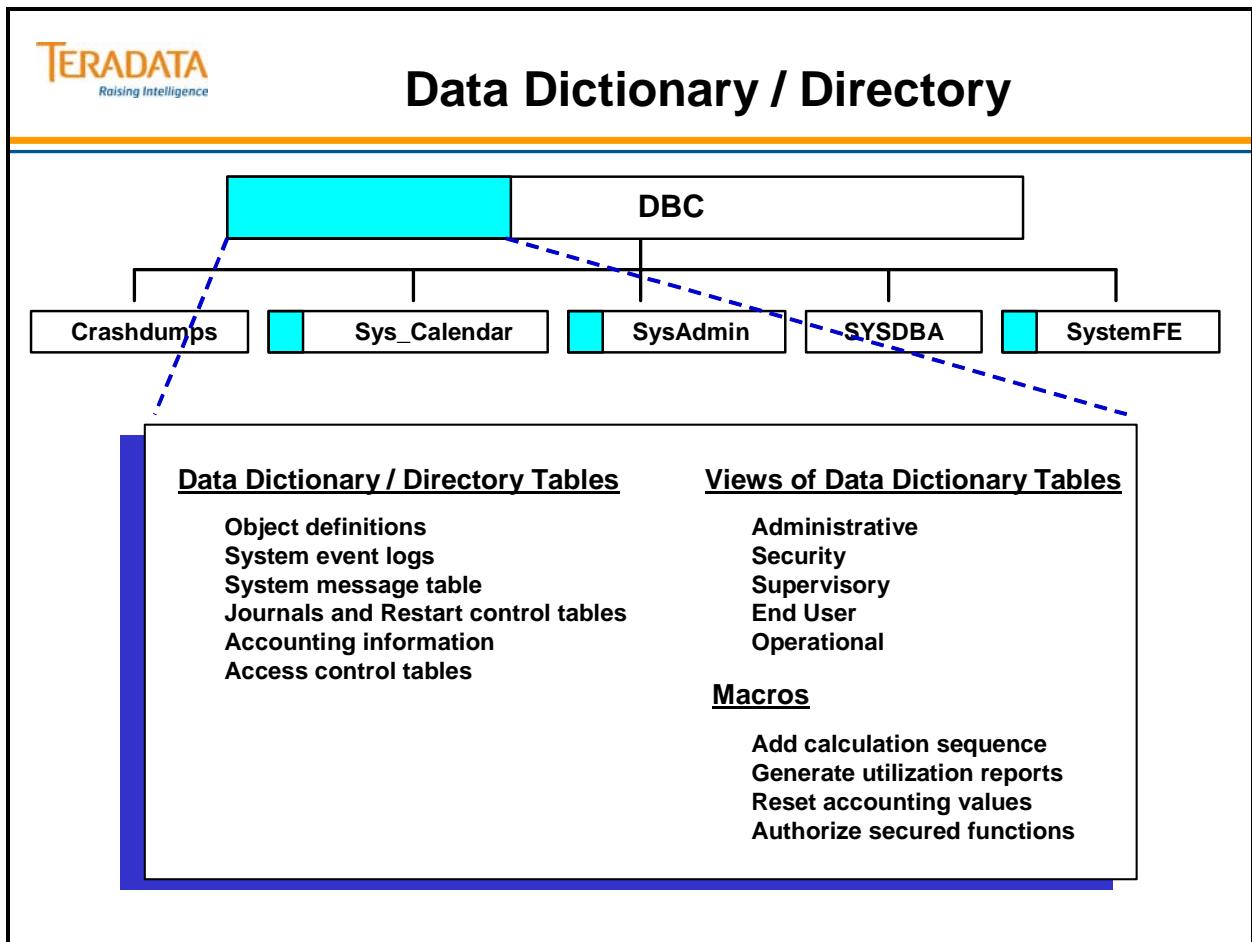
Data dictionary tables are present when you install the system.

The system references some of these tables with SQL requests, while others are used for system or data recovery only.

Data dictionary views reference data dictionary tables. The system views and macros are created by running the Database Initialization Program (DIP) scripts. When a system is first installed, the **start dip** utility is executed by the installation person/team.

Data dictionary tables are used to:

- Store definitions of objects you create (e.g., databases, tables, indexes, etc.).
- Record system events (e.g., logon, console messages, etc.).
- Hold system message texts.
- Control system restarts.
- Accumulate accounting information.
- Control access to data.



Fallback Protected Data Dictionary Tables

Most data dictionary tables are fallback protected.

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

Note: Every system database and user includes a dummy table named “ALL” (with an internal TableID of binary zeros). This table represents all the tables in a system database or user when, for example, privileges are granted or disk space is summarized at the database level.



Fallback Protected Data Dictionary Tables

AccessRights Users Rights on objects	AccLogRuleTbl Specifies events to be logged	AccLogTbl Logged User-Object events	Accounts Account Codes by user
ALL <i>(Dummy) Represents all tables</i>	ConstraintNames	DBase Database and User Profiles	DBCInfoTbl Software Release & Version
ErrorMsgs Message Codes and text	EventLog Session logon/logoff history	Hosts To override default char. sets	IdCol Maintains Identity column data
Indexes Defines indexes on tables	LogonRuleTbl Users Rights on objects	Next Internal ID for next create	OldPasswords Encoded password history
Owners Hierarchy (Downward)	Parents Hierarchy (Upward)	Profiles Users and logon attributes	RCCConfiguration Archive/Recovery Config
RCEvent Archive/Recovery events	RepGroup (V2R6) Replication Groups for Tables	ResUsage Resource Usage tables	ReferencedTbls Referential Integrity (PK)
ReferencingTbls Referential Integrity (FK)	RoleGrants Users/Roles assigned to Roles	Roles Defined Roles	SessionTbl Current logon information
SW_Event_Log Database Console Log	SysSecDefaults Logon security options	TVFields Table/View column description	TVM Tables, Views and Macros
TableConstraints Table Constraints	TempTables Materialized Temporary Tables	Translation National Character Support	TriggersTbl Stores trigger information

(Partial list of Data Dictionary tables in Teradata 12.0)

Non-Hashed Data Dictionary Tables

The data dictionary tables on the following page contain rows that are *not* distributed using hash maps.

Rows in these tables are stored AMP-locally. For example, the DBC.TransientJournal table rows are stored on the same AMP as the row being modified.

Note: User-defined table rows are *always* hash distributed ... either with or without a fallback copy.

TERADATA
Raising Intelligence

Non-Hashed Data Dictionary Tables

Acctg <i>Resource usage by user/account</i>	ChangedRowJournal <i>Down-AMP Recovery Journal</i>	DatabaseSpace <i>Database and Table space accounting</i>
LocalSessionStatus <i>Last request status by AMP</i>	LocalTransactionStatus <i>Last TXN Consensus status</i>	OrdSysChngTable <i>Table-level recovery</i>
RecoveryLockTable <i>Recovery session locks</i>	RecoveryP JTable <i>Permanent Journal recovery</i>	SavedTransactionStatus <i>AMP recovery table</i>
SysRcvStatJournal <i>Recovery, reconfig, startup information</i>	TransientJournal <i>Back out uncommitted transactions</i>	UtilityLockJournalTable <i>Host Utility Lock records</i>

AMP Cluster

```

graph TD
    AMP1[AMP] --- C1(( ))
    AMP2[AMP] --- C2(( ))
    AMP3[AMP] --- C3(( ))
    AMP4[AMP] --- C4(( ))
    C1 --- RL1[AMP LOCAL ROW]
    C2 --- RL2[AMP LOCAL ROW]
    C3 --- RL3[AMP LOCAL ROW]
    C4 --- RL4[AMP LOCAL ROW]
    
```

The diagram illustrates an AMP Cluster. It features four AMP (Application Message Processor) units, each represented by a green rectangular box labeled "AMP". These units are connected to a single large cylinder representing a row. The cylinder is divided into four sections, each labeled "AMP LOCAL ROW". The first section is labeled "PRIMARY ROW", while the other three are labeled "FALLBACK ROW".

Updating Data Dictionary Tables

Whenever you submit a data definition or data control statement, Teradata system software automatically updates data dictionary tables.

When you use the EXPLAIN modifier to describe a DDL statement, you can view updates to the data dictionary tables.

The EXPLAIN modifier is a helpful function that allows you to understand what happens when you execute an SQL statement.

- The statement is not executed.
- The type of locking used is described.
- At least five different tables are updated when you define a new table.



Updating Data Dictionary Tables

```
EXPLAIN CREATE TABLE Orders
  ( order_id      INTEGER NOT NULL,
    order_date    DATE FORMAT 'yyyy-mm-dd',
    cust_id       INTEGER )
  UNIQUE PRIMARY INDEX (order_id);
```

12.0 Explain

1) First, we lock TFACT.Orders 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 read 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 read on a RowHash for deadlock prevention, and we lock a distinct DBC."pseudo table" for write on a RowHash for deadlock prevention.

3) We lock DBC.ArchiveLoggingObjsTbl 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.Indexes for write on a RowHash, we lock DBC.DBase for read on a RowHash, and we lock DBC.AccessRights for write on a RowHash.

4) We execute the following steps in parallel.

- 1) We do a single-AMP ABORT test from DBC.ArchiveLoggingObjsTbl by way of the primary index.
- 2) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
- 3) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index.
- 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 TFACT.Orders.

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.

Supplied Data Dictionary Views

System views are supplied from the data dictionary for frequently used data. The system views do not contain data. They are stored as entries in the data dictionary until you submit an SQL statement that uses them. Views of data dictionary tables are provided for the same reasons that views are defined for any database application.

Data dictionary table column names are re-titled and formatted. Derived values are computed from data dictionary tables. Most supplied views reference more than one table and have the join syntax included. Supplied views also allow you, as the database administrator, to limit access to data dictionary information and provide a consistent image of the data stored in the data dictionary. In practice, as the administrator you may grant permission to the appropriate members of your organization to use any supplied view.

The installation script for the standard views is contained in the supplied Database Initialization Program (DIP) scripts. There are three ways that DIP can be executed:

- From the DBW Supervisor screen, as a utility with the command: **START DIP**
- From a UNIX command line with the command: **/tpasw/bin/dip**
- Use BTEQ and execute each script individually: **.run <name of script>**

The DIP installation screen lists each DIP script separately. You are given the option of choosing “All” to execute all the DIP scripts with one command, or you can choose and run each script separately.

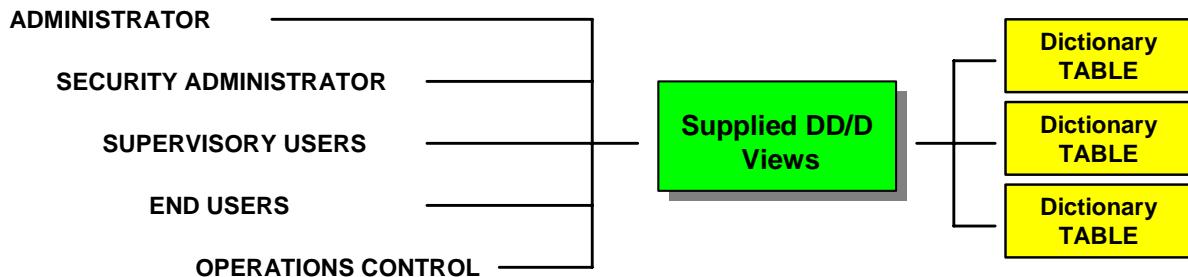
There are specific DIP scripts that can be executed to enable specific functions. For example, the **dipacc** script will create the DBC.AccLogRules macro and will allow access logging to be turned on. Access logging will be discussed in a later module.

When DIP is executed, the following menu (e.g., V2R6.1) is provided:

- | | |
|--------------|-----------------------------|
| 1. DIPERR | - Error Messages |
| 2. DIPRSS | - ResUsage Tables |
| 3. DIPVIEW | - System Views |
| 4. DIPOHL | - Online Help |
| 5. DIPSYSFE | - System FE Macros |
| 6. DIPACR | - Access Rights |
| 7. DIPCRASH | - Crashdumps Database |
| 8. DIPRUM | - ResUsage Views and Macros |
| 9. DIPCAL | - Calendar Tables/Views |
| 10. DIPCCS | - Client Character Sets |
| 11. DIPOCES | - Cost Profiles |
| 12. DIPUDT | - UDT Cast Macros |
| 13. DIPALL | - All of the Above |
| 14. DIPACC | - Access Logging |
| 15. DIPPATCH | - Stand-alone Patches |



Supplied Data Dictionary Views



- **Views are representations of data accessed/derived from DD/D tables.**
 - Clarify tables – re-title tables and/or columns; reorder and format columns, etc.
 - Simplify operations – supply join operation syntax; select and project relevant rows and columns.
 - Limit access to data – exclude certain rows and/or columns from selection.
- **View definitions are stored in DBC.TVM.**
- **View column information is stored in DBC.TVFields.**
- **DIP scripts install the dictionary views.**

Restricted Views

There are two versions of the system views: restricted [x] and non-restricted [non-x]. The system administrator can load either or both versions.

Non-X views are named according to the contents of their underlying tables. DiskSpace, TableSize, and SessionInfo are examples of Non-X views.

X Views are the same views with an appended WHERE clause. The WHERE clause limits the information returned by a view to only those rows associated with the requesting user.

Granted Rights

By default, the SELECT privilege is granted to PUBLIC User on most views in X and non-X versions. This privilege allows any user to retrieve view information via the SELECT statement. The system administrator can use GRANT or REVOKE statements to grant or revoke a privilege on any view to or from any user at any time.

Special Needs Views

Some views are applicable only to users who have special needs. For example, the administrator, a security administrator, or a Teradata field engineer may need to see information that other users do not need. Access to these views is granted only to the applicable user.

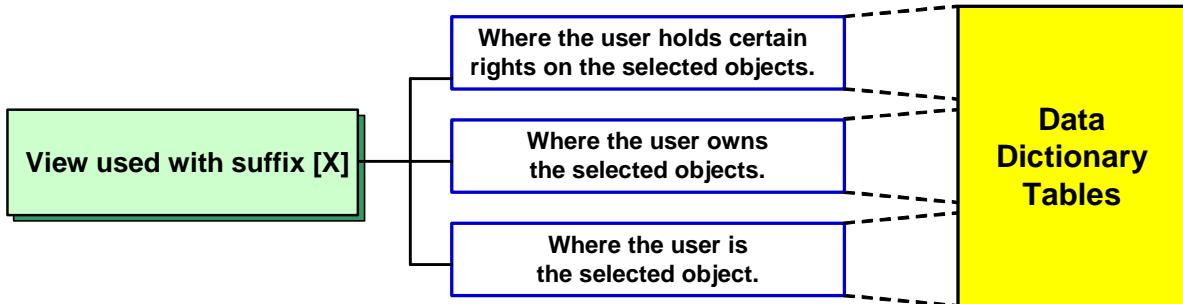
Access Tests

Limited 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. It can take longer to receive a response when a user accesses a restricted view.

Restricted Views

These views limit the scope of what a user can access in the DD/D.

Views with an [X] suffix typically make the following three tests before returning information to the user:



For example,

- The following query returns information about ALL parents and children.
`SELECT Child, Parent FROM DBC.Children;`
- The restricted [x] version of this view selects only information on objects owned by the executing user or that the requesting user has access to:
`SELECT Child, Parent FROM DBC.ChildrenX;`

Suffix Options with Views

The following are the view forms:

- Without the X (for example, DBC.AccountInfo and DBC.AccountInfoV), they display global information.
- With the X (for example, DBC.AccountInfoX and AccountInfoVX), they display information associated with the requesting user only.
- With the V (for example, AccessLogV), they display information associated with the Unicode version, where object name columns have a data type of VARCHAR(128).
- Without the V (for example, DBC.AccountInfo or DBC.AccountInfoX), they display information associated with the Compatibility version, where object name columns have a data type of CHAR(30).

Operations that use restricted views tend to take longer to run because these views access more data dictionary tables. By contrast, operations that use unrestricted views may run faster but return more rows.

To control access to data dictionary information, you can grant users permission to access only restricted views.



Suffix Options with Views

Views may optionally include a suffix of [V] and/or [X].

- Views without the V are the "Compatibility" version of the views, where object name columns have a data type of CHAR(30).
- In general, the X version is for restricted views.
- Starting with Teradata 12.0, most views will have a "V" version which is the newer version of the view. This version of a view has object name columns with a data type of VARCHAR(128). This version can display information associated with Unicode.

For example:

- **DBC.AccountInfo** – displays global information using CHAR(30)
- **DBC.AccountInfoV** – displays global information using VARCHAR(128)
- **DBC.AccountInfoX** – displays information associated with the requesting user only using CHAR(30).
- **DBC.AccountInfoVX** – displays information associated with the requesting user only using VARCHAR(128).

Selecting Information about Created Objects

The following views return information about created objects:

Note: The table “Indexes” is referenced by a view spelled “Indices.”

Object Definition System Views

View Name	Data Dictionary Table	Purpose
DBC.Children[V][X]	DBC.Owners	Provides information about hierarchical relationships.
DBC.Databases[V][X]	DBC.DBase	Provides information about databases, users and their immediate parents.
DBC.Users[V]	DBC.DBase	Similar to DataBases view, but includes columns specific to users.
DBC.Tables[V][X]	DBC.TVM	Provides data about tables, views, macros, triggers, and stored procedures.
DBC.ShowTblChecks[V][X]	DBC.TableConstraints	Database table constraint information.
DBC.ShowColChecks[V][X]	DBC.TVFields	Information about columns in tables and views, and parameters in macros.
DBC.Columns[V][X]	DBC.TVFields	Data about columns in tables and views as well as parameters in macros.
DBC.Indices[V][X]	DBC.Indexes	Data about indexes on tables.
DBC.IndexConstraints[V][X]	DBC.DBase DBC.TableConstraints	Provides information about index constraints implied by a partitioning expression
DBC.AllTempTables[V][X]	DBC.TempTables	Information about all global temporary tables materialized in the system.
DBC.Triggers[V][X]	DBC.TriggersTbl	Information about event-driven triggers attached to a single table and stored in the database.

Note: X views on DBC.ShowTblChecks, DBC.ShowColChecks, DBC.IndexConstraints, and DBC.Triggers are new for V2R6.0.



Selecting Information about Created Objects

DBC.Children[V][X]	Hierarchical relationship information.
DBC.Databases[V][X]	Database, user and immediate parent information.
DBC.Users[V]	Similar to Databases view, but includes columns specific to users.
DBC.Tables[V][X]	Tables, views, macros, triggers, and stored procedures information.
DBC.ShowTblChecks[V][X]	Database table constraint information.
DBC.ShowColChecks[V][X]	Database column constraint information.
DBC.Columns[V][X]	Information about columns/parameters in tables, views, and macros.
DBC.Indices[V][X]	Table index information.
DBC.IndexConstraints[V][X]	Provides information about index constraints, e.g., PPI definition.
DBC.AllTempTables[V][X]	Information about global temporary tables materialized in the system.
DBC.Triggers[V][X]	Information about event-driven, specialized procedures attached to a single table and stored in the database.

Children View

The Children view lists the names of databases and users and their parents in the hierarchy.

<u>Column</u>	<u>Definition</u>
Child	Name of a child database or user
Parent	Name of a parent database or user

Example

The diagram on the facing page uses an SQL statement to list the parents of the current user. The SQL keyword USER causes the parser to substitute the “User Name” of the user who has logged on and submitted the statement. The results of the request show one child, tfact02, and three parents.



Children View

Provides the names of all databases, users and their owners.

DBC.Children[V][X]

Child	Parent
-------	--------

Example:

Using the unrestricted form of the view and a WHERE clause, list your parents.

```
SELECT      *
FROM        DBC.Children
WHERE       CHILD = USER;
```

Example Results:

Child	Parent
tfact02	DBC
tfact02	Teradata_Factory
tfact02	Sysdba

Databases View

The Databases view returns information about databases and users from the DBC.DBase table.

Notes:

- Only the *immediate* owner is identified in this view. Use the parent column of the Children view to select *all* owners.
- The data dictionary records the name of the creator of a system user or database, as well as the date and time the user created the object. This information is not used by the software, but is recorded in DBC.DBase for historical purposes.

Column definitions in this view include:

<u>Column</u>	<u>Definition</u>
OwnerName	The IMMEDIATE parent (owner)
ProtectionType	Default protection type for tables created within this database: F = Fallback N = No Fallback
JournalFlag	Two characters (before and after image) where: S = Single D = Dual N = None L = Local For Example: SD = Single before, dual after image NL = (Single) Local after image
CreatorName	Name of the user who created the object.
CreateTimeStamp	Date and time the user created the object.

Example

The SQL request on the facing page uses the Databases view to find the users with the names that start with TFACT and identify the creator, permanent disk space limit, and database type.



Databases View

Provides information about databases and users.

DBC.Databases[V][X]

DatabaseName	CreatorName	OwnerName	AccountName
ProtectionType	JournalFlag	PermSpace	SpoolSpace
TempSpace	CommentString	CreateTimeStamp	LastAlterName
LastAlterTimeStamp	DBKind	AccessCount **	LastAccessTimeStamp **

** Optional use starting with V2R5.1

Example:

For databases/users with a name of "tfact%", find the creator name, when it was created, its max perm space, and the type (database or user).

```
SELECT DatabaseName (CHAR(10)) AS "Name"
      ,CreatorName (CHAR(10)) AS "Creator"
      ,CreateTimeStamp
      ,PermSpace (FORMAT 'zzz,zz9,999')
      ,DBKind
FROM DBC.Databases
WHERE DatabaseName LIKE 'TFACT%'
ORDER BY 1;
```

Example Results:

Name	Creator	CreateTimeStamp	PermSpace	DBKind
TFACT	DBC	2008-01-16 19:56:21	20,000,000	D
tfact01	Sysdba	2008-01-17 08:06:52	10,000,000	U
tfact02	Sysdba	2008-01-17 08:06:55	10,000,000	U
tfact03	Sysdba	2008-01-17 08:06:57	10,000,000	U

Users View

The Users view is a subset of the Databases view and:

- Limits rows returned from DBC.DBase to only USER rows (e.g., where there is a password).
- Restricts rows returned to:
- The current users' information.
- Information about owned users or databases (i.e., children).
- Information about users on which the current user has DROP USER or DROP DATABASE rights.
- Date and time a user is locked due to excessive erroneous passwords and the number of failed attempts since the last successful one.

The view features CreatorName and CreateTimeStamp columns that display the user name who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

Note: The DBC.Users view is already a restricted view; there is no [X] version.

Column definitions in this view include:

<u>Column</u>	<u>Definition</u>
PermSpace	Maximum permanent space available for this user.
SpoolSpace	Maximum spool space available for this user.
DefaultCollation	A = ASCII E = EBCDIC M = Multinational H = Host (default) C = CharSet_Col J = JIS_Coll

Example:

The SQL statement on the facing page finds the user's default account code, name of his or her immediate owner, and spool space limit.



Users View

Provides information about the users that the requesting user owns or to which he or she has modify rights. This is a restricted view ... there is no [x] version.

DBC.Users[V]

UserName	CreatorName	PasswordLastModDate	PasswordLastModTime
OwnerName	PermSpace	SpoolSpace	TempSpace
ProtectionType	JournalFlag	StartupScript	DefaultAccount
DefaultDatabase	CommentString	DefaultCollation	PasswordChgDate
LockedDate	LockedTime	LockedCount	TimeZoneHour
TimeZoneMinute	DefaultDateFormat	CreateTimeStamp	LastAlterName
LastAlterTimeStamp	DefaultCharType	RoleName (V2R5)	ProfileName (V2R5)
AccessCount **	LastAccessTimeStamp **		

** Optional use starting with V2R5.1

Example:

Find your default account code, the name of your immediate owner, and max spool space.

```
SELECT    UserName
          ,DefaultAccount
          ,OwnerName
          ,SpoolSpace      (FORMAT 'zzz,zz9,999')
        FROM    DBC.Users
        WHERE   UserName = USER;
```

Example Results:

UserName	DefaultAccount	Owner	SpoolSpace
tfact02	\$M_9038_&D&H	Teradata_Factory	50,000,000

Tables View

The Tables view accesses the data dictionary table, DBC.TVM, which contains descriptions of tables, views, macros, journals, join indexes, triggers, and stored procedures.

The view features a TableKind column that allows you to specify the *kind* of object to reference. The view also features CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

The PrimaryKeyIndexID column identifies the columns used as the primary index. As the administrator, use this view to find NO FALBACK tables (where ProtectionType = 'N').

Additional column definitions for this view include:

<u>Column</u>	<u>Definition</u>
Version	A number incremented each time a user alters a table.
RequestText	Returns the text of the most recent DDL statement that was used to CREATE or MODIFY the table.
TableKind	T = Table M = Macro V = View G = Trigger P = Stored Procedure F = User-defined Function I = Join index N = Hash Index J = Journal table

The SQL statement on the facing page requests a list of all tables, views and macros that contain the letters “rights” in their name. The response displays the database name, table name, and a code for the type of object.

With Teradata V2R6, three new columns were added to this view.

RepStatus – identifies the replicated table status for the table. It is NULL if the table is not a member of any replication group.

UtilVersion – contains the utility version count. This column is modified to match the Version column when a significant change of the table definition occurs that would prohibit an incremental restore or copy of an archive.

QueueFlag – this field specifies the queue option as a single character whose value can be either Y if it is queue table, or N if it is not a queue table.



Tables View

Provides information about tables, views, macros, journals, join indexes, hash indexes, triggers, and stored procedures.

DBC.Tables[V][X]

DataBaseName	TableName	Version	TableKind
ProtectionType	JournalFlag	CreatorName	RequestText
CommentString	ParentCount	ChildCount	NamedTblCheckCount
UnnamedTblCheckExist	PrimaryKeyIndexId	RepStatus (V2R6)	CreateTimeStamp
LastAlterName	LastAlterTimeStamp	RequestTxtOverFlow	AccessCount **
LastAccessTimeStamp **	UtilVersion (V2R6)	QueueFlag (V2R6)	

** Optional use starting with V2R5.1

Example:

List the objects that contain the characters "rights" in their name.

```
SELECT TRIM(DatabaseName) || '.' || TableName AS "Qualified Name"
      ,TableKind
FROM DBC.Tables
WHERE TableName LIKE '%rights%'
ORDER BY 1, 2 ;
```

Example Results:

Qualified Name	TableKind
DBC.AccessRights	T
DBC.AllRights	V
DBC.AllRoleRights	V
DBC.UserGrantedRights	V
DBC.UserRights	V
DBCUserRoleRights	V

Columns View

The Columns view returns information from the DBC.TVFields table.

This data dictionary table includes information about:

- Table and view columns
- Macro and stored procedure parameters

Like several other views in this module, the Columns view features CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

As an administrator, you may use this view to enforce domain constraints. The following SELECT statement provides an example:

```
SELECT      DatabaseName, TableName  FROM DBC.TVFields
WHERE       ColumnTitle LIKE 'amount'
AND        ColumnType NE 'D'
OR         DecimalTotalDigits NE 7
OR         DecimalFractionalDigits NE 2
ORDER BY   1,2;
```

Some of the common column types for this view include:

I = INTEGER	F = FLOAT	BF = BYTE Fixed	CV = VARCHAR
I1 = BYTEINT	DA = DATE	BV = BYTE Variable	BO = BLOB
I2 = SMALLINT	D = DECIMAL	CF = CHARACTER Fixed	CO = CLOB
I8 = BYTEINTEGER	GF = GRAPHIC	GV = VARGRAPHIC	UT = UDT Type
TS = TIMESTAMP	TZ = TIME w ZONE	SZ = TIMESTAMP w ZONE	

Example

The SQL statement on the facing page displays selected parameters for the “ResCPUbyAMP” macro.

Miscellaneous Notes:

- The SPPParameterType field specifies the type of the parameter in case of stored procedure object as I (in), O (out) and B (inout).
- The UpperCaseFlag field indicates whether the column is to be stored in uppercase and whether comparisons on the column are case specific. U = Uppercase and not specific, N = not uppercase and not specific, C = not uppercase and specific.
- The CompressValueList field contains the list of values that will be compressed from the column.
- ColumnUDTName (V2R6) – this field specifies the name of a UDT if that column data type is a UDT (User Defined Type). Created for a future Teradata release.



Columns View

Provides information about columns in tables and views, and parameters in macros and stored procedures.

DBC.Columns[V][X]

DatabaseName	TableName	ColumnName	ColumnFormat
ColumnTitle	SPPParameterType	ColumnType	UDTColumnName (V2R6)
ColumnLength	DefaultValue	Nullable	CommentString
DecimalTotalDigits	DecimalFractionalDigits	ColumnId	UpperCaseFlag
Compressible	CompressValue	ColumnConstraint	ConstraintCount
CreatorName	CreateTimeStamp	LastAlterName	LastAlterTimeStamp
CharType	IdColType	AccessCount **	LastAccessTimeStamp **
CompressValueList			

** Optional use starting with V2R5.1

Example:

Use this view to show the parameters of the DBC.ResCPUbyAMP macro.

```
SELECT ColumnName, ColumnFormat, DefaultValue
FROM DBC.Columns
WHERE DatabaseName = 'DBC' AND TableName = 'ResCPUbyAMP' ;
```

Example Results:

ColumnName	ColumnFormat	DefaultValue
FromDate	YYYY-MM-DD	Date
FromNode	X(6)	'000-00'
FromTime	99:99:99	0.000000000000E000
ToTime	99:99:99	9.99999000000000E005
ToDate	YYYY-MM-DD	Date
ToNode	X(6)	'999-99'

Indices View

The Indices view returns information about each indexed column from the DBC.Indexes table. (A compound index returns multiple rows.) Use the view to list tables with non-unique primary indexes (NUPI). (These tables may be subject to skewed data distribution.)

The view includes CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

The following SELECT statement shows an example of how to list NUPI tables:

```
SELECT DatabaseName, TableName
FROM DBC.Indices
WHERE IndexType = 'P'
AND UniqueFlag = 'N'
ORDER BY 1, 2;
```

Column definitions for this view include:

Column	Definition
IndexType	P = Primary (non-partitioned table) Q = Primary (partitioned table) S = Secondary U = Unique constraint (USI with NOT NULL) K = Primary Key J = Join V = Value-ordered secondary H = Hash-ordered ALL (covering) secondary O = Value-ordered ALL (covering) secondary I = Ordering column of a composite secondary index M = Multi-Column Statistics D = Derived column partition statistics
UniqueFlag	Y = Unique N = Non-unique
Column Position	Position of a column within an index. This value may be greater than one if the column is part of a composite index. 1 = Field 1 column of a join index 2 = Field 2 column of a join index

Example

The example on the facing page shows a data request about the Employee_Phone table indices in the current database. The result displays three rows. Notice that the secondary index is a composite index consisting of two columns.

Miscellaneous Note:

IndexMode is H (secondary index rows are hash distributed to the AMPs), L (secondary index rows are on the same AMP as the referenced data row), or NULL (primary index). If the index type is J or N, index mode is L but has no meaning.



Indices View

Provides information about each indexed column defined for each table.

DBC.Indices[V][X]

DatabaseName	TableName	IndexNumber	IndexType
UniqueFlag	IndexName	ColumnName	ColumnPosition
CreatorName	CreateTimeStamp	LastAlterName	LastAlterTimeStamp
IndexMode	AccessCount **	LastAccessTimeStamp **	

** Optional use starting with V2R5.1

Example:

Select information about the Employee Phone table indices in the current database.

```

SELECT      ColumnName (CHAR(15)) AS "Column Name"
           ,UniqueFlag          AS "Unique"
           ,IndexType            AS "Type"
           ,IndexName             AS "Name"
           ,IndexNumber           AS "IndNo"
           ,ColumnPosition        AS "ColPos"
FROM        DBC.Indices
WHERE       TableName = 'Emp_Phone'
AND         DatabaseName = DATABASE
ORDER BY   IndNo, ColPos ;
  
```

Example Results:

Column Name	Unique	Type	Name	IndNo	ColPos
employee_number	N	P	?	1	1
area_code	N	S	ac_phone	4	1
phone_number	N	S	ac_phone	4	2

Indices View (Second Example)

Column definitions for this view include:

<u>Column</u>	<u>Definition</u>
IndexType	P = Primary S = Secondary U = USI (U is used if USI is created via UNIQUE constraint) J = Join V = Value-ordered secondary H = Hash-ordered ALL (covering) secondary O = Value-ordered ALL (covering) secondary I = Ordering column of a composite secondary index
UniqueFlag	Y = Unique N = Non-unique
Column Position	Position of a column within an index. This value may be greater than one if the column is part of a composite index. 1 = Field 1 column of a join index 2 = Field 2 column of a join index

In previous releases, the IndexType and UniqueFlag were both needed to identify a USI.

Starting with V2R5, an IndexType of U indicates a USI that is created via a UNIQUE constraint. However, a USI that is created as a UNIQUE INDEX in the table definition is still identified via the IndexType and UniqueFlag.

Example:

The example on the facing page shows a data request about the Department table indices in the current database. Notice that this example contains a join index.



Indices View (Second Example)

Example:

Select information about the Department table indices in the current database.

```

SELECT   ColumnName (CHAR(15)) AS "Column Name"
        ,UniqueFlag          AS "Unique"
        ,IndexType            AS "Type"
        ,IndexName             AS "Name"
        ,IndexNumber           AS "IndNo"
        ,ColumnPosition        AS "ColPos"
FROM     DBC.Indices
WHERE    TableName = 'Department'
AND      DatabaseName = DATABASE
ORDER BY IndNo, ColPos ;
  
```

Example Results:

Column Name	Unique	Type	Name	IndNo	ColPos
dept_number	Y	P	?	1	1
dept_name	Y	U	?	4	1
dept_number	N	J	Dept_Jnlx	8	1
dept_name	N	J	Dept_Jnlx	8	2
dept_mgr_number	N	J	Dept_Jnlx	8	3

SQL of how this Join Index was created:

```

CREATE JOIN INDEX Dept_Jnlx AS SELECT      dept_number, dept_name, dept_mgr_number
                                     FROM        Department
                                     PRIMARY INDEX (dept_mgr_number);
  
```

IndexConstraints View

DBC.IndexConstraints is a new system view (V2R5) and it provides information about index constraints, specifically a partitioning expression constraint. A partitioning expression is an implied index constraint.

A ConstraintType = Q indicates a partitioned primary index.

The ConstraintText indicates the partitioning constraint. The general format of the text will be:

```
CHECK ( (<partitioning-expression>) BETWEEN 1 AND <max> )
```

<max> is 65535 or the number of partitions defined by the partitioning expression if the partitioning expression consists solely of a RANGE_N or CASE_N function.

The definition of the Sales_History table is:

```
CREATE SET TABLE DS.Sales_History, NO FALBACK,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL
(
  store_id INTEGER NOT NULL,
  item_id INTEGER NOT NULL,
  sales_date DATE FORMAT 'YYYY-MM-DD',
  total_revenue DECIMAL(9,2),
  total_sold INTEGER,
  note VARCHAR(256) CHARACTER SET LATIN NOT CASESPECIFIC)
UNIQUE PRIMARY INDEX ( store_id ,item_id ,sales_date )
PARTITION BY RANGE_N(sales_date BETWEEN
DATE '2001-01-01' AND DATE '2006-12-31' EACH INTERVAL '1' MONTH );
```

An example of constraint text for the Sales_History table is:

```
CHECK ((RANGE_N("sales_date" BETWEEN
  DATE '2001-01-01' AND DATE '2006-12-31' EACH INTERVAL '1' MONTH ))
  BETWEEN 1 and 65535)
```



IndexConstraints View

Provides information about partitioned primary index constraints.

This view only displays tables with an index constraint type of "Q".

- Q indicates a table with a PPI

DBC.IndexConstraints[V][X]

DatabaseName	TableName	IndexName	IndexNumber
ConstraintType	ConstraintText	ConstraintCollation	CollationName
CreatorName	CreateTimeStamp		

Example:

List all of the partitioning expression constraints for all tables in the current database.

```
SELECT      TableName          AS "Table Name"
           ,ConstraintText    AS "Constraint Text"
FROM        DBC.IndexConstraints
WHERE       DatabaseName = DATABASE;
```

Example Results:

Table Name	Constraint Text
Sales_History	CHECK ((RANGE_N("sales_date" BETWEEN ...
Store_Sales	CHECK ((store_id) BETWEEN 1 and 65535)
Store_Item	CHECK (((store_id - 1000)* 1000) + (item_id - ...
Store_Revenue	CHECK ((CASE_N(total_revenue < 2000, ...

ShowTblChecks View

The ShowTblChecks view displays database table constraint information. The view features CreatorName and CreateTimeStamp columns that display the name of the user who created an object, and the date and time he or she created it. The LastAlterName and LastAlterTimeStamp columns list the name of the last user to modify an object, as well as the date and time.

Column definitions for this view include:

<u>Column</u>	<u>Definition</u>
DatabaseName	Names of databases that contain tables with table-level checks.
TableName	Table names that have table-level constraints.
CheckName	Table-level check name.
TblCheck	Returns unresolved text for the table-level check condition.
CreatorName	Name of the user who created the object.
CreateTimeStamp	The time the object was created.

The SQL to create these table level constraints follows:

```
ALTER TABLE Employee ADD CONSTRAINT Emp_Chk1
CHECK (Employee_number >= 100000);
```

```
ALTER TABLE Department ADD CONSTRAINT Dept_Chk1
CHECK (Dept_number >= 1000);
```

```
ALTER TABLE Job
ADD CHECK (Job_code >= 3000);
```

Note: If a check constraint is created at the column level and it is a named constraint, then it will appear in this view.

```
CREATE SET TABLE TFACT.Dept2 ,
( dept_number INTEGER NOT NULL
CONSTRAINT Dept_chk2 CHECK (dept_number >= 1000),
dept_name CHAR(20) NOT NULL UNIQUE,
dept_mgr_number INTEGER,
budget_amount DECIMAL(10,2) )
UNIQUE PRIMARY INDEX ( dept_number );
```



ShowTblChecks View

Provides information about check constraints at the table level and “named” column constraints.

DBC.ShowTblChecks[V][X]

DatabaseName CreatorName	TableName CreateTimeStamp	CheckName	TblCheck
-----------------------------	------------------------------	-----------	----------

Example:

Display table constraint information.

```
SELECT      TableName (CHAR(10))
           ,CheckName (CHAR(10))
           ,TblCheck
FROM        DBC.ShowTblChecks
WHERE       DatabaseName = 'TFACT';
```

Example Results:

TableName	CheckName	TblCheck
DEPARTMENT	Dept_Chk1	CONSTRAINT "Dept_Chk1" CHECK ("Dept_nu
EMPLOYEE	Emp_Chk1	CONSTRAINT "Emp_Chk1" CHECK ("Employe
JOB	?	CHECK ("Job_code" >= 3000)

Note: The first two are named constraints and the third is an unnamed constraint. All three of these constraints were created at the table level.

ShowColChecks View

The ShowColChecks view displays database column constraint information for unnamed constraints. The view features CreatorName and CreateTimeStamp columns that display the name of the user who created an object and the date and time he or she created it.

Column definitions for this view include:

<u>Column</u>	<u>Definition</u>
DatabaseName	Names of databases that contain tables with table-level checks.
TableName	Table names that have table-level constraints.
ColumnName	Names of columns that contain column-level check constraints.
ColCheck	Returns unresolved text for the column-level check condition.
CreatorName	Name of the user who created the object.
CreateTimeStamp	The time the object was created.

The SQL to create these column level constraints follows:

```

CREATE SET TABLE TFACT.Emp_2
(employee_number INTEGER NOT NULL CHECK (employee_number >= 100000),
 dept_number      INTEGER,
 :
 salary_amount DECIMAL(10,2)
UNIQUE PRIMARY INDEX ( employee_number );

CREATE SET TABLE TFACT.Dept_2
(dept_number INTEGER NOT NULL      CHECK (dept_number >= 1000),
 :
 budget_amount DECIMAL(10,2)
UNIQUE PRIMARY INDEX ( dept_number );

CREATE SET TABLE TFACT.Job_2
(job_code INTEGER NOT NULL          CHECK (Job_code >= 3000),
 job_desc CHAR(20) NOT NULL UNIQUE)
UNIQUE PRIMARY INDEX ( job_code );

```



ShowColChecks View

Provides information about unnamed column check constraints.

[DBC.ShowColChecks\[V\]\[X\]](#)

DatabaseName CreatorName	TableName CreateTimeStamp	ColumnName	ColCheck
-----------------------------	------------------------------	------------	----------

Example:

Show information about column constraints for a database.

```
SELECT      TableName      (CHAR(10))
           ,ColumnName      (CHAR(10))
           ,ColCheck
FROM        DBC.ShowColChecks
WHERE       DatabaseName = 'TFACT';
```

Example Results:

TableName	ColumnName	ColCheck
EMP_2	employee_number	CHECK ("employee_number" >= 100000)
DEPT_2	dept_number	CHECK ("dept_number" >= 1000)
JOB_2	job_code	CHECK ("job_code" >= 3000)

Note: A second set of Employee, Department, and Job tables were created with unnamed CHECK constraints at the column level.

Triggers View

The Triggers view provides information about event-driven, specialized procedures attached to a single table and stored in the Teradata database.

Using Triggers

Characteristics of triggers include:

- To define a trigger, use the CREATE TRIGGER statement.
- To cause the database to execute a trigger, use the INSERT, UPDATE or DELETE statements on the specified table or view.
- There are two kinds of triggers:
- Row triggers (R) evaluate each row changed by the trigger action.
- Statement triggers (S) evaluate the entire statement.
- When a triggered statement fires a trigger, cascading ensues that, in some instances, can fire other triggers and become triggering statements.
- Use the REFERENCING clause when you reference subject tables that are qualified with old or new table values. In addition, all subject table columns must use new or old correlation names.

Note: A positioned (updateable cursor) UPDATE or DELETE is not allowed to fire a trigger and generates an error. In addition, the FastLoad and MultiLoad utilities return an error if you have any triggers enabled on the target table.

Column definitions for this view include:

<u>Column</u>	<u>Definition</u>
ActionTime	Indicates when the triggered action fires. B = Before trigger statement changes the table. A = After trigger statement changes the table.
Event	Indicate which of the following SQL statements fires the trigger: U = UPDATE I = INSERT D = DELETE
Column Position	Position of a column within an index. This value may be greater than one if the column is part of a composite index.



Triggers View

Provides information about event-driven triggers attached to a single table and stored in the database.

DBC.Triggers[V][X]

DatabaseName	SubjectTableDataBaseName	TableName
TriggerName	EnabledFlag	ActionTime
Event	Kind	OrderNumber
TriggerComment	RequestText	CreatorName
CreateTimeStamp	LastAlterName	LastAlterTimeStamp
AccessCount **	LastAccessTimeStamp **	CreateTxtOverflow

** Optional use
starting with
V2R5.1

Example:

Show if the trigger is enabled, when it fires, the type of statement that fires it, and the kind.

```
SELECT      TableName          (CHAR(10)) AS TName
           ,TriggerName        (CHAR(12))
           ,EnabledFlag         AS "Enabled"
           ,ActionTime          AS Action
           ,Event
           ,Kind
FROM        DBC.Triggers
WHERE       DatabaseName = DATABASE;
```

Example Results:

TName	TriggerName	Enabled	Action	Event	Kind
Employee	Raise_Trig	Y	A	U	R

AllTempTables View

This view provides information about all temporary tables materialized in the system.

Global Temporary Tables

Use global temporary tables to store temporary, immediate results from multiple queries into working tables. To create a global temporary table, you must state the keywords GLOBAL TEMPORARY in the CREATE TABLE statement. The temporary table defined during the CREATE TABLE statement is referred to as the base temporary table.

When referenced in an SQL session, a local temporary table is materialized with the exact same definition as the base table. Once the temporary table is materialized, subsequent DML statements referring to that table are mapped to the materialized instance.

Note: After you create a global temporary table definition, use the INSERT statement to create a local instance of the global temporary table to use during the session.

Temporary versus Permanent Tables

Temporary tables are different than permanent tables in the following ways:

- They are always empty at the start of a session.
- Their contents cannot be shared by other sessions.
- You can empty them at the end of each transaction.
- The system automatically drops them at the end of each session.



AllTempTables View

Provides information about all global temporary tables materialized in the system.

DBC.AllTempTables[V][X]

HostNo B_TableName	SessionNo E_TableID	UserName	B_DatabaseName
-----------------------	------------------------	----------	----------------

Example:

Show all temporary tables materialized in the system.

```
SELECT HostNo
      ,SessionNo
      ,UserName (CHAR(10))
      ,B_DatabaseName AS "DataBase"
      ,B_TableName AS "Table Name"
   FROM DBC.AllTempTables;
```

Example Results:

HostNo	SessionNo	UserName	Database	Table Name
01	20887	TFACT02	PD	GT_DEPTSALARY
01	20908	TFACT01	PD	GT_DEPTSALARY

Referential Integrity Views

The facing page identifies a number of views that can be used to list tables with referential integrity and the state of the referential integrity on the tables.

Additional views that specifically provide referential integrity information are listed below. The effects of referential integrity on the database can be seen in the series of views identified with the letters “RI”. The X views were implemented in Teradata V2R6.0.

<u>VIEW NAME</u>	<u>DESCRIPTION</u>
DBC.All_RI_Children[V][X]	This view shows the Referential Integrity Constraints defined in the database, from the Child-Parent point of view.
DBC.All_RI_Parents[V][X]	This view shows the same information as the above view, but from the Parent-Child perspective.
DBC.RI_Distinct_Children[V][X]	Provides information about tables in child-parent order without the duplication that could result from multi-column foreign keys.
DBC.RI_Distinct_Parents[V][X]	Provides information about tables in parent-child order without the duplication that could result from multi-column foreign keys.
DBC.RI_Child_Tables[V][X]	Provides information about tables in child-parent order. It is similar to the All_RI_Children view but returns the internal IDs of databases, tables, and columns.
DBC.RI_Parent_Tables[V][X]	Provides information about all tables in parent-child order. It is similar to the All_RI_Parents view but returns the internal IDs of databases, tables, and columns instead of names.



Referential Integrity Views

Views that can be used to provide information about Referential Integrity are:

View

Description

DBC.Tables[V][X]

Can be used to list parent and child counts for tables.

DBC.All_RI_Children[V][X]

Information about tables in **child-parent** order. Also identifies if RI constraint is **consistent** or **inconsistent**.

DBC.All_RI_Parents[V][X]

Similar to above view but provides Information in **parent-child** order.

DBC.Databases2[V][X]

Can be used to identify child tables with **unresolved reference constraints**.

Note: Additional RI views are listed on the facing page.

Using the DBC.Tables View

The DBC.Tables[X] views (described previously) provide parent and child counts for tables within a database.

The examples on the following pages are based on the following CREATE and ALTER TABLE statements.

```

CREATE SET TABLE Employee
( employee_number      INTEGER      NOT NULL      PRIMARY KEY
,dept_number           INTEGER
,emp_mgr_number       INTEGER
,job_code              INTEGER
,last_name             CHAR(20)
,first_name            VARCHAR(20)
,salary_amount         DECIMAL(10,2) ) ;

CREATE SET TABLE Department
( dept_number          INTEGER      NOT NULL      PRIMARY KEY
,dept_name              CHAR(20)      NOT NULL      UNIQUE
,dept_mgr_number        INTEGER
,budget_amount          DECIMAL (10,2) ) ;

CREATE SET TABLE Job
( job_code              INTEGER      NOT NULL      PRIMARY KEY
,job_desc                CHAR(20)      NOT NULL      UNIQUE ) ;

CREATE SET TABLE Emp_Phone
( employee_number        INTEGER      NOT NULL
,area_code                SMALLINT      NOT NULL
,phone_number              INTEGER      NOT NULL
,extension                 INTEGER
,PRIMARY KEY (employee_number, area_code, phone_number) )
PRIMARY INDEX (employee_number);

ALTER TABLE Employee      ADD CONSTRAINT emp_dept_ref
FOREIGN KEY (dept_number) REFERENCES
Department (dept_number);

ALTER TABLE Employee      ADD CONSTRAINT emp_job_ref
FOREIGN KEY (job_code) REFERENCES
Job (job_code);

ALTER TABLE Employee ADD CONSTRAINT emp_mgr_ref
FOREIGN KEY (emp_mgr_number) REFERENCES
Employee (employee_number);

ALTER TABLE Department     ADD CONSTRAINT dept_mgr_ref
FOREIGN KEY (dept_mgr_number) REFERENCES
Employee (employee_number);

ALTER TABLE Emp_Phone     ADD CONSTRAINT phone_emp_ref
FOREIGN KEY (employee_number) REFERENCES
Employee (employee_number);

```



Using the DBC.Tables View

The DBC.Tables[V][X] views can be used to list parent and child counts for tables.

- **ParentCount** – how many foreign keys does a table have or how many parents does the table reference?
- **ChildCount** – how many foreign keys reference this table or how many children does the table have?

Example:

List the tables objects in the database TFACT and identify parent and child counts.

```
SELECT      TableName
           ,TableKind
           ,ParentCount
           ,ChildCount
           ,DatabaseName
           ,TableKind
FROM        DBC.Tables
WHERE       DatabaseName = 'TFACT'
AND         TableKind = 'T'
ORDER BY    2, 1 ;
```

Example Results:

TableName	TableKind	ParentCount	ChildCount
Department	T	1	1
Employee	T	3	3
Emp_Phone	T	1	0
Job	T	0	1
Salary_Log	T	0	0

Referential Integrity States

The facing page identifies three states that are associated with a Referential Integrity constraint.



Referential Integrity States

The status of a Referential integrity constraint is classified as follows:

- **Unresolved reference constraint** – the FK exists, but the PK does not.
 - Creating a table with a FK before creating the table with Parent Key (PK).
 - Restoring a table with a FK and the table with the PK does not exist or hasn't been restored.
- **Inconsistent reference constraint** – both the FK and the PK exist, but the constraint is marked as inconsistent.
 - When either the child or parent table is restored, the reference constraint for the child table is marked as inconsistent.
 - no inserts, updates, deletes or table changes are allowed
- **Consistent reference constraint** – both the FK and the PK exist and are considered consistent, but FK values without a corresponding PK value are identified as "**invalid rows**".
 - Typically occurs when the reference constraints are created on already populated tables or ...
 - Following a REVALIDATE REFERENCES command after a restore of either the child or parent table.

DBC.All_RI_Children View

The DBC.All_RI_Children[X] views provide information about all tables in child-parent order.

A table can have many referential constraints defined. When either the child or parent table is restored, these constraints are marked ***inconsistent***. The DBC.All_RI_Children view provides information about reference constraints.

The REVALIDATE REFERENCES FOR statement validates these inconsistent constraints against the target table.

If inconsistent constraints remain after a REVALIDATE REFERENCES FOR statement has been executed, the SQL statement ALTER TABLE DROP INCONSISTENT REFERENCES must be used to remove them.

REVALIDATE REFERENCES FOR creates error tables containing information about data rows that failed referential constraint checks.



DBC.ALL_RI_Children View

This view is used to identify tables with RI in child-parent order and can also be used to show if the RI constraint is consistent or **inconsistent**.

DBC.ALL_RI_Children
[V][X]

IndexID	IndexName	ChildDB	ChildTable
ChildKeyColumn	ParentDB	ParentTable	ParentKeyColumn
InconsistencyFlag	CreatorName	CreateTimeStamp	

Example:

```
SELECT IndexID (FORMAT 'z9') AS ID
      ,IndexName ,ChildTable ,ChildKeyColumn
      ,ParentTable ,ParentKeyColumn ,InconsistencyFlag AS ICF
   FROM DBC.ALL_RI_Children
 WHERE ChildDB = 'PD' ORDER BY 3, 4 ;
```

Results:

ID	IndexName	ChildTable	ChildKeyColumn	ParentTable	ParentKeyColumn	ICF
0	dept_mgr_ref	Department	dept_mgr_number	Employee	employee_number	N
0	emp_dept_ref	Employee	dept_number	Department	dept_number	Y
8	emp_mgr_ref	Employee	emp_mgr_number	Employee	employee_number	Y
4	emp_job_ref	Employee	job_code	Job	job_code	Y
0	phone_emp_ref	Emp_Phone	employee_number	Employee	employee_number	N

Options to handle inconsistent references:

- **ALTER TABLE tname DROP INCONSISTENT REFERENCES;** – FK constraints are dropped – use the ALTER TABLE command to create new references constraints.
- Use the **REVALIDATE REFERENCES** command (ARC facility).

DBC.Databases2 View

The DBC.Databases2[V][X] views provide ID definition information about databases and provide a count of unresolved reference constraints for any tables within the database.

It is similar to the Databases view but returns the ID of the database and Referential Integrity (RI) information instead of the other information (creator name, owner name, etc.) provided by the Databases view.

You can control who has access to internal ID numbers by limiting the access to the Databases2 view while allowing more users to access the names via the Databases view.

Example

The SQL request on the facing page uses the Databases2 view to find databases that have tables with unresolved references.

The columns selected are:

DatabaseName	Returns the name of a database with the indicated count of unresolved references.
DatabaseId	Returns the ID of the database with the indicated count of unresolved references.
UnresolvedRICount	Returns the total number of unresolved Referential Integrity (RI) constraints in the database. If one table has 2 unresolved reference constraints, then the count is 2.



DBC.Databases2 View

The DBC.Databases2[V][X] views provide information about **unresolved reference constraints**.

Unresolved reference constraints are caused by:

- Creating a table with a Foreign Key before creating the table with Parent Key (Primary Key).
- Restoring a table with a Foreign Key and the Parent Key (Primary Key) table does not exist or hasn't been restored.

DBC.Databases2

DatabaseName	DatabaseID	UnresolvedRCount
--------------	------------	------------------

Example: List all databases with unresolved references.

```
SELECT DatabaseName
      ,DatabaseId
      ,UnresolvedRCount
  FROM DBC.Databases2
 WHERE UnresolvedRCount > 0;
```

Results:

DatabaseName	DatabaseID	UnresolvedRCount
PD	0000FE03	2

Restore the Parent Table to “resolve” the references constraint, but the references constraint is marked as inconsistent.

Time Stamps in Data Dictionary

The Teradata Database includes a time stamp in most of the data dictionary tables.

The time stamp feature is meant to facilitate and enhance your system administration tasks by providing a means to identify obsolete objects, and clean up and recapture space. Time stamps also help you determine when a change to an object occurred for system maintenance activities and problem investigations.

The facing page shows the time stamp fields in dictionary tables, system views and dictionary views. A description of these fields follows.

Time Stamp Field Definitions:

Create Time Stamp	The time the object was created in ANSI TimeStamp Format.
CreateUID	User ID of the user who created the object.
CreatorName	Name of the user who created the database, table, or the name of the user's creator.
LastAlterName	Name of the user who last updated the object.
LastAlterTimeStamp	The time the object was last updated in ANSI TimeStamp format.
LastAlterUID	User ID of the user who last updated the object.
AccessCount	The number of times the object was accessed. (Only used if requested.)
LastAccessTimeStamp	The time the object was last accessed in ANSI TimeStamp format.



Time Stamps in Data Dictionary

The Teradata Database features a time stamp in the Data Dictionary tables.

- CreateTimeStamp, CreatorName, LastAlterTimeStamp, LastAlterName

This feature can help system administration tasks by providing a means to identify objects recently updated, obsolete objects, etc. and who altered the objects.

Example:

List all tables that have been altered in January of 2008.

```
SELECT TRIM(DatabaseName) || '.' || TableName
      ,LastAlterName      AS "User Name"
      ,LastAlterTimeStamp AS "Last Alter Date & Time"
FROM   DBC.Tables
WHERE  EXTRACT(YEAR FROM LastAlterTimeStamp) = 2008
AND    EXTRACT(MONTH FROM LastAlterTimeStamp) = 1
ORDER BY 1, 2 ;
```

Example Results:

Qualified Name	User Name	Last Alter Date & Time
HR_Tab.Job	tfact03	2008-01-23 08:10:14
HR_Tab.Location	tfact03	2008-01-23 17:19:07
HR_Tab.Raise_Trig	Sysdba	2008-01-20 04:58:14
HR_Tab.Salary_Log	Sysdba	2008-01-20 04:58:14
TFACT.New_Sales	tfact03	2008-01-24 16:25:27

Teradata Administrator – List Columns of a View

Teradata Administrator can be used to perform many of the functions described in this module.

The facing page shows a simple example of using Teradata Administrator to view the columns in a system view.

TERADATA
Raising Intelligence

Teradata Administrator List Columns of a View

Teradata Administrator can be used to list the columns of DD/D views (and tables).

Appendix C of this manual contains a listing of all the DD/D views and columns.

Name	Type	AccessCount	Last Access	CreatorName
61 60^ CostProfiles_v	View			DBC
62 60^ CostProfileTypes_v	View			DBC
63 60^ CostProfileValues_v	View			DBC
64 60^ CSPSESSIONINFO	View			DBC
65 60^ CSPSESSIONINFOV	View			The CSP
66 60^ Databases	View			DBC
67 60^ Databases2	View			DBC
68 60^ Databases2V	View			DBC
69 60^ Databases2X	View			DBC
70 60^ Databases2X	View			DBC
71 60^ DatabasesV	View			DBC
72 60^ DatabasesVX	View			DBC
73 60^ DatabasesX	View			DRC

List Columns - DBC.Databases

ColumnName	Description
1 DatabaseName	The Databases.DatabaseName field identifies the data base.
2 CreatorName	The Databases.CreatorName field identifies the username that created the database.
3 OwnerName	The Databases.OwnerName field identifies the data base from which this database was created.
4 AccountName	The Databases.AccountName field identifies the account to be charged for the database.
5 ProtectionType	The Databases.ProtectionType field specifies the fallback option default for the database.
6 JournalFlag	The Databases.JournalFlag field specifies the default journal options for the database.
7 PermSpace	The Databases.PermSpace field specifies the total permanent space in bytes.
8 SpoolSpace	The Databases.SpoolSpace field specifies the maximum spool space permitted for the database.
9 TempSpace	The Databases.TempSpace field specifies the maximum temporary space permitted for the database.
10 CommentString	The Databases.CommentString field contains any user-supplied text for the database.
11 CreateTimeStamp	

Teradata Administrator – Object Options

Teradata Administrator can be used to perform a wide range of functions. The facing page shows an example of the options available at a Table object level.

To use the viewing functions of Teradata Administrator, you must have Select access to the DBC views of the Teradata Database. To use the Copy, Drop, Create or Grant options, you must have the corresponding privilege on the table or database that you are trying to modify or create. To use the Browse or Row Count features you must have select access to the Table or View.

TERADATA
Raising Intelligence

Teradata Administrator Object Options

Teradata Administrator can also be used to display object details.

For example, right-click on the object (e.g., Department table) and a menu of options is displayed.

In this example, the Indexes option was selected.

IndexName	ColumnName	Type	Unique	IndexNumber	ColumnPosition
1	Dept_Number	Primary	Y	1	1
2	Dept_Name	Unique Constraint	Y	4	1
3	Dept_Mgr_Number	Secondary	N	8	1

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- The data dictionary consists of tables, views, and macros stored in system user DBC.
- The Teradata Database software automatically updates data dictionary/directory tables as you create or drop objects.
- You can access data dictionary tables with supplied views.
- Data dictionary tables keep track of all created objects:
 - Database and users
 - Tables, views, macros, triggers, stored procedures, and user-defined functions
 - Columns and indexes
 - Hierarchies
- Note: To access information about individual objects stored in the data dictionary, use the HELP and SHOW commands.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. True or False. The DBC.Databases view only contains information about databases; users are not included in this view.
2. True or False. The DBC.Users view only contains information about users; databases are not included in this view.
3. True or False. Queries that use restricted views usually take less time to execute than queries that use unrestricted views.
4. True or False. All of the data dictionary tables are Fallback protected.
5. If a child table exists and the parent table doesn't, the reference constraint is marked as _____.
 - A. Inconsistent
 - B. Unresolved
 - C. Missing
 - D. Invalid
6. After executing the ALTER TABLE ... ADD FOREIGN KEY ... statement, Foreign Key values that are missing in the parent table are marked in an error table and are known as _____ rows.
 - A. Inconsistent
 - B. Unresolved
 - C. Missing
 - D. Invalid

Lab Exercise 42-1

The following pages describe the tasks for this lab exercise.

Notes about DBC.DBCInfo columns:

- The Release column provides the PDE release number.
- The Version column provides the Teradata software version number.



Lab Exercises

Lab Exercise 42-1

Purpose

In this lab, you will use BTEQ or SQL Assistant to view information in the data dictionary using various Data Dictionary views (use Appendix C for details on Data Dictionary views).

What you need

SELECT Access to the Data Dictionary views.

Tasks

1. Using the DBC.DBCInfo view, find the release and version of the system on which you are logged on:

Release (PDE) _____ Version (Teradata) _____

2. Using the DBC.Children view, list your parents' user names.

3. Using the DBC.Databases view, find your:

Immediate parent's name _____

Creator's name _____

Default account code _____

Perm space limit _____

Spool space limit _____

Temp space limit _____

Lab Exercise 42-1 (cont.)

The following pages describe the tasks for this lab exercise.



Lab Exercises

Lab Exercise 42-1 (cont.)

Tasks

4. Using the DBC.Users view, find your:

Default database name _____
Default collation sequence _____
Default date format _____
Create time stamp _____
Last password modification date _____

OPTIONAL: SHOW this view. Note the WHERE conditions. (Remember, this is a restricted view, even though it does not have an [X] suffix.)

5. Using the DBC.Tables view, find the number of tables in the DD/D (user DBC) that are:

Fallback protected _____
Not Fallback protected _____

Modify the query to find the number of tables OTHER THAN DD/D (not DBC tables) that are:

Fallback protected _____
Not Fallback protected _____

Lab Exercise 42-1 (cont.)

The following pages describe the tasks for this lab exercise.



Lab Exercises

Lab Exercise 42-1

Tasks

6. Using the DBC.Columns view, find the number of columns in the entire system defined with *default values*:

Number of columns _____

OPTIONAL: Modify the query to find the number of tables or views that have columns defined with *default values*:

Number of tables _____

7. Using the DBC.IndexConstraints view, list the tables in your database that have a PPI and display the partitioning constraints.

Number of PPI tables _____ What type of constraint does Orders_PPI_M have? _____

8. Using the DBC.Indices view, find the number of tables OTHER THAN Dictionary tables that have non-unique primary indexes (NUPI):

Number of tables _____

Lab Exercise 42-2

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To populate a table:

```
INSERT INTO empty_tablename
SELECT * FROM non_empty_tablename;
```

To grant an Access Right on a table:

```
GRANT REFERENCES ON tablename TO username;
```

To grant an Access Right on a specific column:

```
GRANT REFERENCES (column_name) ON tablename TO username;
```



Lab Exercises

Lab Exercise 42-2

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to establish References constraints between 4 populated tables and view the associated data dictionary entries.

What you need

Populated PD tables and empty tables in your database

Tasks

1. Use INSERT/SELECT to place all rows from the populated PD tables into your empty tables. Verify the number of rows in your tables.

PD.Employee	to populate	Employee	Count = _____
PD.Department	to populate	Department	Count = _____
PD.Job	to populate	Job	Count = _____
PD.Emp_Phone	to populate	Emp_Phone	Count = _____

2. Use the GRANT statement to GRANT yourself the REFERENCES access rights on the tables.

GRANT REFERENCES ON *tablename* TO *username*;

Lab Exercise 42-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To create a References constraint:

```
ALTER TABLE          child tablename
  ADD CONSTRAINT    constraint name
    FOREIGN KEY     (child name)
      REFERENCES    parent tablename (parent column);
```

To select from the DBC.ALL_RI_Children view:

```
SELECT      Indexid      (FORMAT 'z9') AS ID
            ,IndexName
            ,ChildTable
            ,ChildKeyColumn
            ,ParentTable
            ,ParentKeyColumn
  FROM        DBC.ALL_RI_Children
 WHERE       ChildDB = USER
 ORDER BY    1;
```



Lab Exercises

Lab Exercise 42-2 (cont.)

Tasks

3. Create a References constraint between the Employee.Dept_Number column and the Department.Dept_Number column.

What is the name of the Employee RI error table? _____

How many rows are in this table? _____

Which department is not represented in the department table? _____

4. Use the DBC.All_RI_Children view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? _____

Lab Exercise 42-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

Use the following SQL for this exercise.

To create a References constraint:

```
ALTER TABLE          child tablename
  ADD CONSTRAINT    constraint name
    FOREIGN KEY     (child name)
      REFERENCES    parent tablename (parent column);
```

To select from the DBC.ALL_RI_Children view:

```
SELECT      Indexid      (FORMAT 'z9') AS ID
            ,IndexName
            ,ChildTable
            ,ChildKeyColumn
            ,ParentTable
            ,ParentKeyColumn
  FROM        DBC.ALL_RI_Children
 WHERE       ChildDB = USER
 ORDER BY    1;
```



Lab Exercises

Lab Exercise 42-2 (cont.)

Optional Tasks

5. Create the References constraint between the Employee.Job_code column and the Job.Job_Code column.

What is the name of the Employee RI error table? _____

How many rows are in this table? _____

Which job code is not represented in the job table? _____

6. Use the DBC.All_RI_Children view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? _____

7. Create the References constraint between the Employee.Emp_Mgr_Number column and the Employee.Employee_Number column.

What is the name of the Employee RI error table? _____

How many rows are in this table? _____

Which employee does not have a manager (Emp_Mgr_Number is 0 or NULL)? _____

8. Use the DBC.All_RI_Children view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? _____

Teradata Training

Notes

Module 43



Space Allocation and Usage

After completing this module, you will be able to:

- Define permanent space, spool space and operating system space requirements.
- Estimate system capacity.
- Use the AllSpace, DiskSpace, and TableSize views to monitor disk space utilization.
- Use Teradata Administrator to view database and table space utilization.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Permanent Space Terminology	43-4
Spool Space Terminology	43-6
Temporary Space Terminology.....	43-8
Resetting Peak Values.....	43-10
Assigning Perm and Spool Limits	43-12
Giving One User to Another	43-14
Teradata Administrator – Move Space	43-16
Reserving Space for Spool	43-18
Views for Space Allocation Reporting	43-20
DiskSpace View	43-22
TableSize View	43-24
AllSpace View	43-26
DataBaseSpace Table.....	43-28
Different Views — Different Results	43-30
Additional Utilities to View Space Utilization	43-32
Teradata Administrator – Database Menu Options.....	43-34
Teradata Administrator – Object Menu Options.....	43-36
Transient Journal Space	43-38
Ferret Utility.....	43-40
Ferret SHOWSPACE Command	43-42
Ferret SHOWBLOCKS.....	43-44
Ferret SHOWBLOCKS – Subtable Detail.....	43-46
Review Questions	43-48

Permanent Space Terminology

MaxPerm

MaxPerm is the maximum number of bytes available for table, index, and permanent journal storage in a system database or user.

The number of bytes specified is divided by the number of AMP vprocs in the system. The result is recorded on each AMP vproc and may not be exceeded on that vproc. *

Perm space limits are deducted from the limit set for the immediate parent of the object defined.

Perm space is acquired when data is added to a table. The space is released when you delete or drop objects.

CurrentPerm

CurrentPerm is the total number of bytes (including table headers) in use on the database to store the tables, subtables and permanent journals contained in a User or Database. This value is maintained on each AMP vproc.

PeakPerm

PeakPerm is the largest number of bytes ever actually used to store data in a user or database. This value is maintained on each AMP vproc.

Reset the PeakPerm value to zero by using the ClearPeakDisk Macro supplied in User DBC.

Note: Space limits are enforced at the database level. A database or user may own several small tables or a few large tables as long as they are within the MaxPerm limit set on each AMP.

- * Minor exceptions to this rule may occur occasionally. For example, utilities that write data to disk a block at a time (such as FastLoad) check space limits *after* a block is written.



Permanent Space Terminology

MaxPerm

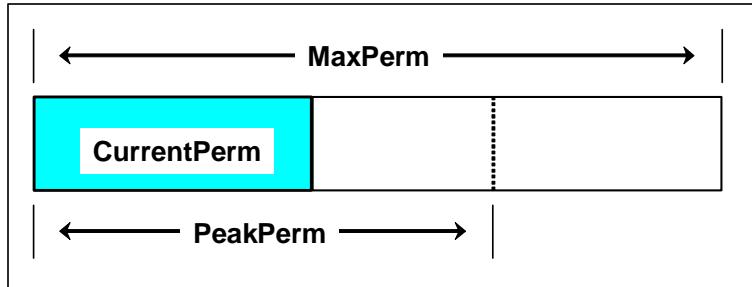
The maximum number of bytes *available* for table, index and permanent journal storage in a database or user.

CurrentPerm

The total number of bytes *in use* to store the tables, subtables, and permanent journals contained in the database or user.

PeakPerm

The *largest* number of bytes actually used to store data in this user since the value was last reset.



Spool Space Terminology

MaxSpool

MaxSpool is a value used to limit the number of bytes the system will allocate to create spool files for a user.

The value you specify may not exceed that of a user's immediate parent (database or user) at the time you create the user. If you do not specify a value, MaxSpool defaults to the parent's MaxSpool value.

Limit the spool space you allocate to users to reduce the impact of "runaway" transactions, such as accidental product joins.

Spool space marked (last use) is recovered by a worker task that is initiated every five minutes.

CurrentSpool

CurrentSpool is the number of bytes in use for running transactions. This value is maintained on each AMP vproc for each user.

PeakSpool

PeakSpool is the maximum number of bytes used by a transaction run for a user since the value was last reset by the ClearPeakDisk Macro (supplied in system user DBC).

Spool Space Terminology

MaxSpool

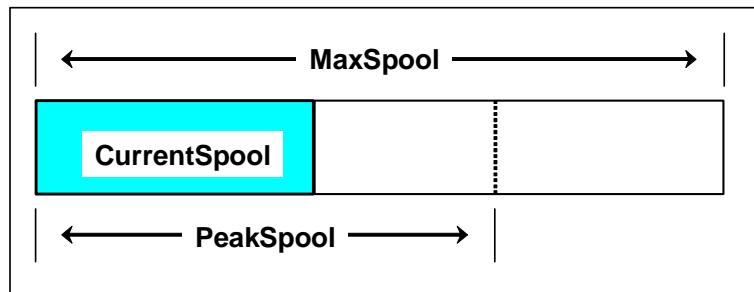
A value used to limit the number of bytes the system will consume to create spool files for a user.

CurrentSpool

The number of bytes currently in use for running transactions.

PeakSpool

The maximum number of bytes used by a transaction run for this user since the value was last reset.



Temporary Space Terminology

MaxTemp

MaxTemp is a value used to limit the number of bytes the system will use to store data for global temporary tables for a user.

The value you specify may not exceed that of a user's immediate parent (database or user) at the time you create the user. If you do not specify a value, MaxTemp defaults to the parent's MaxTemp value.

CurrentTemp

CurrentTemp is the number of bytes in use for global temporary tables. This value is maintained on each AMP vproc for each user.

PeakTemp

PeakTemp is the maximum number of bytes used by global temporary tables for a user since the value was last reset by the ClearPeakDisk Macro (supplied in system user DBC).



Temporary Space Terminology

MaxTemp

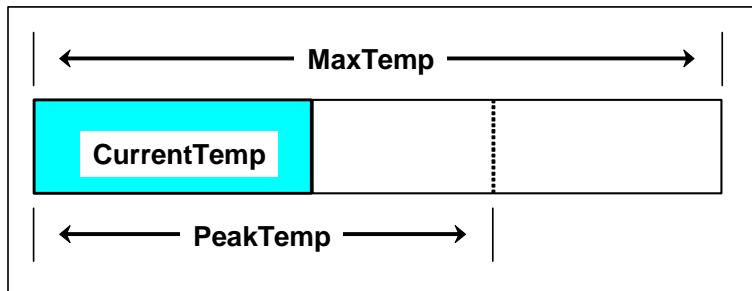
A value used to limit the number of bytes the system will use to store data for global temporary tables for a user.

CurrentTemp

The number of bytes in use for global temporary tables.

PeakTemp

The maximum number of bytes used by global temporary tables for a user since the value was last reset.



Temporary space is released when the user terminates the session or when the user frees the temporary space.

Resetting Peak Values

From time to time, the administrator needs to clear out the values accumulated in the DBC.DataBaseSpace table. These values must be reset to restart the data collection process.

DBC.ClearPeakDisk

The Teradata software provides a macro to reset the PeakPerm, PeakSpool, and PeakTemp values in the DBC.DataBaseSpace table. It may be used to reset the peak values for the next collection period.



Resetting Peak Values

The ClearPeakDisk macro resets PeakPerm, PeakSpool, and PeakTemp values in the DatabaseSpace table.

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

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

This macro may be used to reset peak values for the next data collection period.

To clear accounting values:

```
EXEC DBC.ClearPeakDisk;
```

*** Update completed. 3911 rows changed.

*** Time was 4 seconds.

Assigning Perm and Spool Limits

You define permanent and spool space limits at the database or user level, not at the table level.

When you create databases or users, perm space limits are deducted from the available (unused) space of the immediate owner.

The spool space limit may not exceed that of the immediate owner at the time you create an object. If you do not specify a spool space limit, the new object “inherits” its limit from the immediate owner (user or database).

Example

The diagram on the facing page illustrates how Teradata manages permanent and spool space.

A user, Payroll, has a 25 MB permanent space limit and a 50 MB spool space limit.

Payroll creates two new users, PY01 and PY02. After Payroll creates the new objects, its maximum Perm space drops to 15 MB. PY01 has 6 MB of maximum Perm and PY02 has 4 MB.

Later, Payroll drops user PY02. Payroll’s maximum Perm space increases to 19 MB since it regains the permanent space that used to belong to PY02.

Payroll has a limit of 50 MB of maximum Spool. When it creates PY01, it assigns 35 MB of maximum Spool to the new user. Since there is no statement of spool space for PY02, its maximum Spool defaults to the limit of its immediate parent: 50 MB.

The amount of maximum Perm increases and decreases as the owner creates and drops new users. The spool space figure remains constant even when the owner adds and drops users.



Assigning Perm and Spool Limits

 Payroll

MaxPerm = 25e6 MaxSpool = 50e6

CREATE USER PY01 AS PASSWORD = abc, PERM = 6e6, SPOOL = 35e6;

CREATE USER PY02 AS PASSWORD = xyz, PERM = 4e6;



MaxPerm = 15e6 MaxSpool = 50e6

The maximum Spool value may not exceed that of the immediate owner at the time you create the new user.

What is the maximum Spool limit of PY02?

DROP USER PY02;



MaxPerm = 19e6 MaxSpool = 50e6

The Perm space from PY02 is returned back to the immediate owner.

Giving One User to Another

When you give an object to another object in the hierarchy, all space allocated to that object goes with it. If you drop the object, its space is credited to its immediate owner.

When you give databases or users, all descendants of the given object remain descendants of the given object.

When you give an object to new parents, the ownership of space is transferred; however the limits remain the same.

If you drop an object, its space is credited to its immediate owner.

The facing page illustrates giving a database/user from one database/user to another such as giving Payroll from Human_Resources to Accounting.

Adjusting Perm Space Limits

You can easily adjust perm space limits. Using the illustration on the facing page as an example, you could transfer 10 MB from Human_Resources to Accounting using the following technique:

1. CREATE DATABASE Temp FROM Human_Resources AS PERM = 10000000;
2. GIVE Temp TO Accounting;
3. DROP DATABASE Temp;

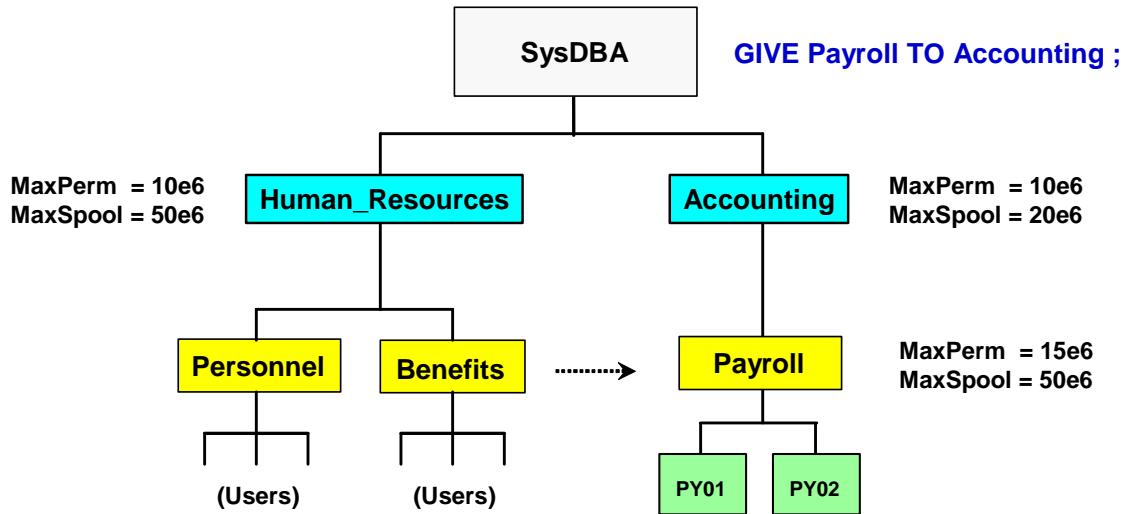
The database TEMP is NOT shown on the facing page, but is used as an example of how space could be transferred from one database to another.

Notes:

You enforce limits when you create an object.

Objects you give may have spool limits that exceed that of their current owner.

Giving One User to Another



- Payroll ownership is transferred (GIVE) to Accounting.
- All descendants (child users/databases, tables, views, etc.) of a “given” object remain with the “given” object.
- The GIVE command may also be used to move Permanent space from one database/user to another database/user.

Teradata Administrator – Move Space

The Move Space function of Teradata Administrator makes it easy to move space from one database/user to another database/user. The Tools > Move Space menu choice displays the Move Space dialog box that you use to reallocate permanent disk space from one database to another:

The example on the facing page illustrates moving 20 MB of Permanent space from the HR_Tab database to the Payroll_Tab database.

The user who has logged onto Teradata Administrator requires the DROP DATABASE access right on HR_Tab and the CREATE DATABASE access right on Payroll_Tab in order to do this move space operation.

As discussed previously, the following commands would accomplish the same thing.

1. CREATE DATABASE Temp FROM HR_Tab AS PERM = 20000000;
2. GIVE Temp TO Payroll_Tab;
3. DROP DATABASE Temp;

The screenshot shows the Teradata Administrator interface with the title "Teradata Administrator – Move Space". On the left, there is a sidebar with sections for "Techniques to move Perm space.", "GIVE command", and "Teradata Administrator". The main area displays a database tree and a table of objects. A blue arrow points from the "Move Space..." option in the Tools menu to a "Move Space [WXP_TD]" dialog box. The dialog box has fields for "From Database" (set to HR_Tab), "To Database" (set to Payroll_Tab), and "Number of Bytes" (set to 20). There are radio buttons for KB, MB, and GB.

Techniques to move Perm space.

- GIVE command
- Teradata Administrator

GIVE command

- 1) CREATE temp database with PERM space.
- 2) GIVE temp database to another database.
- 3) DROP temp database.

Teradata Administrator

- Tools -> Move Space – moves Perm space from one database to another.

Reserving Space for Spool

Spool space serves as temporary storage for returned rows during transactions that users submit. To ensure that space is always available, you may want to set aside about 20 to 25% of total available space as spool space. To do this, you can create a special database called Spool_Reserve. This database will not be used to load tables.

Decision support applications should reserve more of the total disk space as reserved spool space since their SQL statements generate larger spool files. OLTP applications can use less as reserved spool space because their statements generate smaller spool files.

The above actions guarantee that data tables will never occupy more than 75% to 80% of the total disk space. Since there is no data stored in Spool_Reserve, the system will use its permanent space as spool space when necessary.

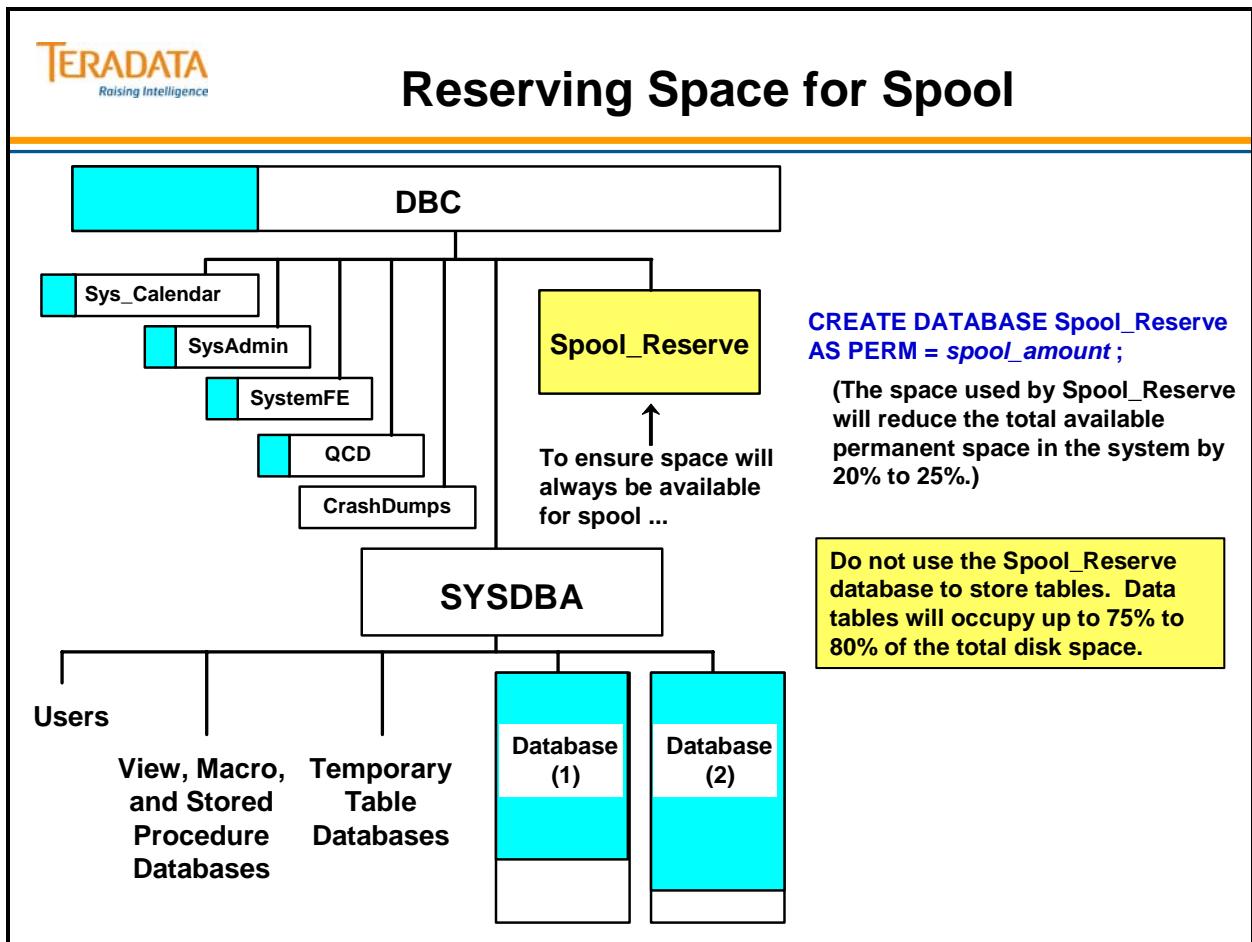
Orphan or Phantom Spool Issues

Spool tables are temporary work tables which are created and dropped as queries are executed. When a query is complete, all of the spool tables it used should be dropped automatically.

Like all tables, spool tables require a Table ID. There is a range of tableids exclusively reserved for spool tables (C000 0001 thru FFFF FFFF) and the system cycles through them. If a table is incorrectly not dropped, it remains in existence. Eventually, the system will cycle through all the table ids and reassign the tableid which is in use by our left over spool table. Usually, the presence of this table is detected, the query which was going to use the tableid is aborted – even though it is innocent of any wrongdoing – and the following message is returned to the user and put in the error log:

*** FAILURE 2667 Left-over spool table found: transaction aborted.

The unusual cases of leftover spool are covered in another module later in this course.



Views for Space Allocation Reporting

Use the following system views to report current space allocation:

DBC.DiskSpace[X]

This view gives AMP vproc information about disk space usage for any database or account. It gets this information from the ALL table.

DBC.TableSize[X]

This view gives AMP vproc information about disk space usage (excluding spool) for any table or account.

DBC.AllSpace[X]

This view gives AMP vproc information about disk space usage (including spool) for any database, table, or account.

Each of these views references the non-hashed DBC.DataBaseSpace table.



Views for Space Allocation Reporting

View Name

Description

DBC.DiskSpace[V][X]

AMP vproc information about disk space usage (including spool) for any database or account.

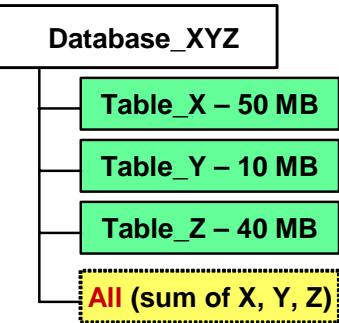
DBC.TableSize[V][X]

AMP vproc information about disk space usage (excluding spool) for any database, table or account.

DBC.AllSpace[V][X]

AMP vproc information about disk space usage (including spool) for any database, table, or account.

Example:



SUM (CurrentPerm) using the 3 views

DBC.DiskSpace

Table All – 100 MB

DBC.TableSize

Table_X – 50 MB

Table_Y – 10 MB

Table_Z – 40 MB

DBC.AllSpace

Table_X – 50 MB

Table_Y – 10 MB

Table_Z – 40 MB

Table All – 100 MB

100 MB

200 MB

DiskSpace View

The DiskSpace[V][X] view provides AMP vproc information about disk space usage at the database level. This view includes spool space usage.

The PeakSpool column can be used to determine the maximum amount of spool space that a user has used (via DatabaseName and AccountName columns).

Example

The SELECT statement on the facing page calculates the percentage of disk space used in the owner's database. The result displays a partial report with five rows. The DS database has the highest percentage of utilized space at 88.41%. SystemFE has the lowest at 12.39%.

Note: In the statement, use NULLIFZERO to avoid a divide exception.



DiskSpace View

Provides AMP Vproc disk space usage at the database level, including spool space.

DBC.DiskSpace[V][X]

Vproc	DatabaseName	AccountName	MaxPerm	MaxSpool
MaxTemp	CurrentPerm	CurrentSpool	CurrentTemp	PeakPerm
PeakSpool	PeakTemp	MaxProfileSpool	MaxProfileTemp	

This view selects values for TableName "ALL" (TableID = '000000000000' XB).

Example:

Calculate the percentage of permanent space used in databases and users.

```
SELECT DatabaseName
      ,CAST (SUM (MaxPerm) AS FORMAT 'zzz,zzz,zz9')
      ,CAST (SUM (CurrentPerm) AS FORMAT 'zzz,zzz,zz9')
      ,CAST (((SUM (CurrentPerm))/
              NULLIFZERO (SUM(MaxPerm))) * 100)
              AS FORMAT 'zz9.99%') AS "% Used"
   FROM DBC.DiskSpace
  GROUP BY 1
 ORDER BY 4 DESC ;
```

Example Results:

DatabaseName	Sum(MaxPerm)	Sum(CurrentPerm)	% Used
DS	209,715,200	185,413,632	88.41%
TFACT	104,857,600	45,396,480	43.29%
AU	20,000,000	3,978,240	19.89%
Sys_Calendar	15,000,000	2,647,040	17.65%
SystemFe	1,000,000	123,904	12.39%

TableSize View

The TableSize view is a Data Dictionary view that provides AMP Vproc information about disk space usage at a table level, optionally for tables the current User owns or has SELECT privileges on.

Example

The SELECT statement on the facing page looks for poorly distributed tables by displaying the CurrentPerm figures for a single table on all AMP vprocs.

The result displays one table, **Table2**, which is evenly distributed across all AMP vprocs in the system. The CurrentPerm figure is nearly identical across all vprocs. The other table, **Table2_nupi**, is poorly distributed. The CurrentPerm figures range from 95,232 bytes to 145,920 bytes on different AMP vprocs.



TableSize View

Provides AMP Vproc disk space usage at table level.

DBC.TableSize[V][X]

Vproc CurrentPerm	DatabaseName PeakPerm	AccountName	TableName
----------------------	--------------------------	-------------	-----------

Excludes TableName "All" (TableID = '000000000000' XB)

Example:

Look at table distribution
across AMPs.

```
SELECT      Vproc
           ,CAST (TableName
                  AS FORMAT 'X(20)')
           ,CurrentPerm
           ,PeakPerm
  FROM        DBC.TableSize
 WHERE       DatabaseName = USER
 ORDER BY    TableName, Vproc ;
```

Vproc	TableName	CurrentPerm	PeakPerm
0	Table2	127,488	253,440
1	Table2	127,488	253,440
2	Table2	127,488	253,952
3	Table2	127,488	253,952
4	Table2	128,000	255,488
5	Table2	128,000	255,488
6	Table2	126,976	251,904
7	Table2	126,976	251,904
0	Table2_nupi	95,232	95,232
1	Table2_nupi	95,232	95,232
2	Table2_nupi	145,920	145,920
3	Table2_nupi	145,920	145,920
4	Table2_nupi	123,904	123,904
5	Table2_nupi	123,904	123,904
6	Table2_nupi	145,408	145,408
7	Table2_nupi	145,408	145,408

AllSpace View

The AllSpace[V][X] view provides AMP vproc information about disk space usage at the database and table level. This information includes the “All” table.

Example

The SELECT statement on the facing page lists the MaxPerm and CurrentPerm figures for each table in the user's space. The result displays three table names: **All**, **Table2**, and **Table2_nupi**.

The “All” table represents all tables that reside in the user's space. The MaxPerm figure for “All” is the amount of permanent space defined for that user. There are only two tables in this user's defined space.

Note: The “Table2 and Table2_nupi” tables display zero bytes in the MaxPerm column.

This is because tables do not have MaxPerm space, only databases and users do, as represented by the “All” table.



AllSpace View

AMP vproc disk space usage at the database AND table level.

DBC.AllSpace[V][X]

Vproc
MaxPerm
CurrentSpool
PeakTemp

DatabaseName
MaxSpool
CurrentTemp

AccountName
MaxTemp
PeakPerm

TableName
CurrentPerm
PeakSpool

Includes TableName "All" (TableID = '000000000000' XB)

Example:

List all tables (by Vproc)
contained in the user's space.

```
SELECT      Vproc
           ,CAST (TableName AS
                  FORMAT 'X(20)')
           ,MaxPerm
           ,CurrentPerm
  FROM        DBC.AllSpace
 WHERE       DatabaseName = USER
 ORDER BY    TableName, Vproc ;
```

Vproc	TableName	MaxPerm	CurrentPerm
0	All	1,250,000	222,720
1	All	1,250,000	222,720
2	All	1,250,000	273,408
3	All	1,250,000	273,408
:	:	:	:
0	Table2	0	127,488
1	Table2	0	127,488
2	Table2	0	127,488
3	Table2	0	127,488
:	:	:	:
0	Table2_nupi	0	95,232
1	Table2_nupi	0	95,232
2	Table2_nupi	0	145,920
3	Table2_nupi	0	145,920
:	:	:	:

DataBaseSpace Table

The DataBaseSpace table tracks and stores information about disk space usage for objects in the Teradata system. The information is updated as users create new databases and add tables to them. The facing page illustrates four columns from DataBaseSpace. The SQL to generate this report follows:

```
SELECT      DatabaseID, TableID, Vproc, CurrentPermSpace
FROM        DBC.DataBaseSpace
WHERE       DatabaseID='00001404'XB
ORDER BY    2;
```

DataBaseSpace Columns

The four columns described below are used by the AllSpace, DiskSpace and TableSize views to produce disk space utilization reports:

DatabaseID

A DatabaseID is a unique identification number assigned to a database when the CREATE DATABASE statement is issued. The SQL statement adds a new row to the DataBaseSpace table and automatically assigns an internal database ID that corresponds with the database name assigned by the user.

TableID

TableID is a unique identification number assigned to a table when the CREATE TABLE statement is issued. The SQL statement adds a new row to the DataBaseSpace table and automatically assigns an internal table ID that corresponds with the table name assigned by the user.

Each database has a table ID 000000000000. This table has a special purpose. It displays the total amount of PermSpace used by the entire database, not just a single table. This table name is referenced as “All”.

Vproc

Vproc is the **logical vproc number** where a table is stored. Since tables are evenly distributed across all AMP vprocs, a single table is stored on several different vprocs.

CurrentPermSpace

CurrentPermSpace is the number of bytes of permanent space taken up on a specific vproc by that table. Table ALL (ID of 000000000000) displays the total amount of permanent space used by the tables stored in that database.



DataBaseSpace Table

Four columns from
DBC.DataBaseSpace:

Notes: 1,019,904

+ 1,020,928

2,040,832

2,040,832

1,019,904

+ 1,020,928

4,081,664

DatabaseID	TableID	Vproc	CurrentPermSpace
UPI			
00001404	00000000000000	0	222,720
00001404	00000000000000	1	222,720
00001404	00000000000000	2	273,408
00001404	00000000000000	3	251,904
00001404	00000000000000	4	251,904
00001404	00000000000000	5	272,384
00001404	00000000000000	6	272,384
00001404	00000000000000	7	
00001404	0000260C0000	0	127,488
00001404	0000260C0000	1	127,488
00001404	0000260C0000	2	127,488
00001404	0000260C0000	3	127,488
00001404	0000260C0000	4	128,000
00001404	0000260C0000	5	1,019,904
00001404	0000260C0000	6	126,976
00001404	0000260C0000	7	126,976
00001404	0000270C0000	0	95,232
00001404	0000270C0000	1	95,232
00001404	0000270C0000	2	
00001404	0000270C0000	3	145,920
00001404	0000270C0000	4	145,920
00001404	0000270C0000	5	1,020,928
00001404	0000270C0000	6	123,904
00001404	0000270C0000	7	145,408
00001404	0000270C0000		145,408

Different Views — Different Results

Conflicting Results

It would seem logical that query results would be the same for any of the preceding views, since they all use the same underlying table. However, query results can differ depending upon which view you select.

The SQL statements and results on the facing page illustrate how a single SQL statement can produce a different result for each view.

For example, when we select the MAX (CurrentPerm) and SUM (CurrentPerm) from each of the AllSpace, DiskSpace, and TableSize views, our results will differ. The SUM (CurrentPerm) value from the DBC.AllSpace view represents the Sum of “All” tables (i.e., the database total) *plus* the sum of *each* table in the database. The results are misleading. We suggest that you do not use this query.

We recommend that you use the DBC.DiskSpace view for queries at the *database* level and use the DBC.TableSize view for queries at the *table* level.

Sum(CurrentPerm)

The DiskSpace view displays 2,040,832 bytes of total permanent space used for database ID 00001404. This figure reflects the total number of bytes stored on each processor in table ID 000000000000 or table ALL. Remember, the DiskSpace view reports on database space usage.

The TableSize view also displays 2,040,832 bytes of total CurrentPerm. This figure comes from the individual tables stored within the same database. The total comes from adding all of the bytes in tables 0000260C0000 and 0000270C0000 together. Since DiskSpace reports on the database and TableSize reports on the individual tables in the database, both result in the same figure.

The AllSpace view displays 4,081,664 bytes which is double what the other two views reported. This view displays the total of all tables including table ID 000000000000 or table ALL. Since table name ALL already contains the totals from all of the other tables, the resulting total is double what it should be.

Maximum(CurrentPerm)

Both DiskSpace and AllSpace display 273,408 bytes as the largest number of permanent space. Both views read the result from table 000000000000. TableSize, on the other hand, displays 145,920. TableSize looks at individual tables. It excludes figures stored in table ID 000000000000 or table ALL.



Different Views — Different Results

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

Maximum (CurrentPerm)	Sum (CurrentPerm)
273,408	2,040,832

Values only represent table ALL.

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

Maximum (CurrentPerm)	Sum (CurrentPerm)
145,920	2,040,832

Values represent all of the actual tables except table ALL.

```
SELECT      MAX(CurrentPerm)
           ,SUM(CurrentPerm)
FROM        DBC.AllSpace
WHERE       DatabaseName = USER ;
```

Maximum (CurrentPerm)	Sum (CurrentPerm)
273,408	4,081,664

Values represent all of the actual tables and table ALL.

Additional Utilities to View Space Utilization

Examples of additional tools that may be used to view database and table space utilization are provided in this module.



Additional Utilities to View Space Utilization

Teradata Administrator – graphical tool to easily view space usage

- Database menu
 - Child Space – space usage for all child databases of the selected database
 - Table space – space usage for all tables of the selected database
- Object menu
 - Space Summary – current and peak perm usage of the specified table
 - Space by AMP – current and peak perm usage of the specified table by AMP

Ferret – system utility started via Database Console or Teradata Manager

- ShowSpace – space usage (perm, spool, and temporary) at the system level
- ShowBlocks – allocation of permanent space at the table and subtable level

Question – Why use ShowBlocks to determine space usage at a table level?

- “How much perm space is currently being used by a secondary index?”
 - This level of detail is not available with DBC.TableSize and Teradata Administrator – only provide current perm space usage at the table level.
 - **ShowBlocks provide subtable space information** – multiply the typical block size times the number of blocks to determine subtable space usage.

Teradata Administrator – Database Menu Options

Use the command selections on the Database pull-down menu to indicate the type of information you want displayed.

A check mark indicates the current setting of your database Default View option (i.e., the information displayed when you double click on a database.)

Note: The Database menu does not appear on the Teradata Administrator menu bar until you establish a connection with a database server.

Select a database from the database tree windowpane and make a selection from the Database pull-down menu. Selections and corresponding information displayed are identified in the table below.

Selection	Display Information
List Tables	Each table in the selected database
List Views	Each view in the selected database
List Macros	Each macro in the selected database
Database Info	The selected database itself
Database Rights	Access rights for each table, view, and macro in the selected database
Table Space	Space usage for each table in the selected database
Child Space	Space usage for each database that is owned directly by the selected database
List Databases	All databases and users created under the selected database
List All DB/TVM	Each table, view, macro, trigger and join index in the selected database and show all Child Database/Users in the tree
Open / Close DB	Expansion or contraction of the database tree



Teradata Administrator Database Menu Options

Teradata Administrator can be used to easily view database/user space usage.

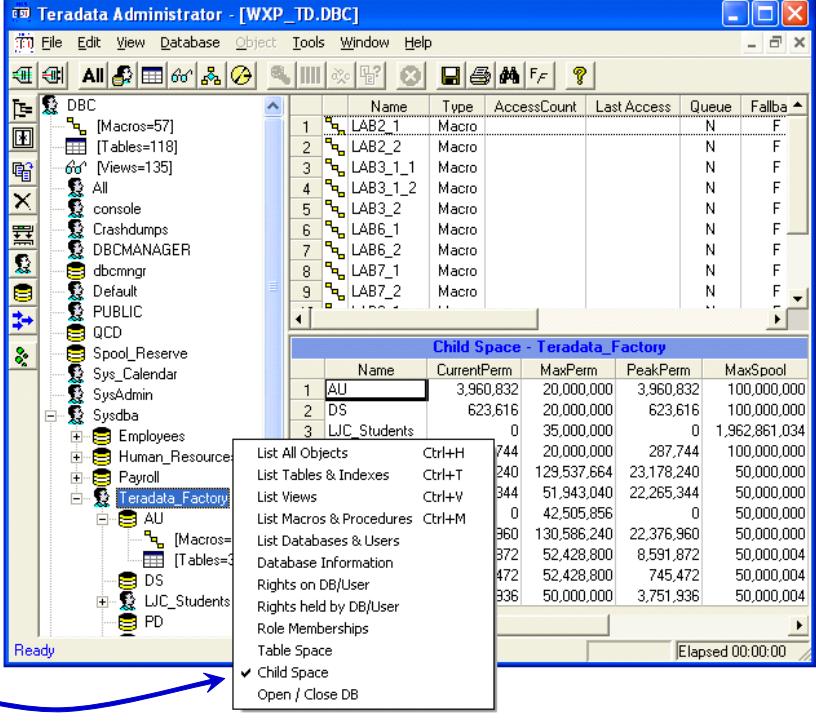
Database menu

- List databases, tables, views and macros.
- Display database information and access rights.
- View table and child space usage.
- Information appears in grid area.

Database > Child Space
displays a space usage report for each database that is owned directly by selected database.

OR

Right-click on a database/user and select Child Space.



The screenshot shows the Teradata Administrator interface with the title bar "Teradata Administrator - [WXP_TD.DBC]". The left pane displays the Database Browser with nodes like DBC, All, and specific databases such as Teradata_Factory, Employees, Human_Resource, Payroll, and LJC_Students. The right pane shows a grid of objects under "DBC" and a detailed "Child Space - Teradata_Factory" report. A context menu is open over the "Teradata_Factory" node, listing options like "List All Objects", "List Tables & Indexes", "List Views", etc., with "Child Space" highlighted with a blue arrow.

Teradata Administrator – Object Menu Options

Select an item in the upper grid area, and use the submenu selections on the Object menu to display detail information, described below, about the selected table, macro, or view.

Object Type	Selection	Display Information
Table, View	List Columns	Information about the columns of the selected table or view
Table	Index	The indexes for the selected table
Table	Statistics	Statistics information for the selected table
Table, View	Row Count	The number of rows in the selected table or view
Table, View	Browse	Information from the data rows of the selected table or view
Table, View, Macro	Info	General information about the selected object
Table	Space Summary	Space usage information for the selected table
Table	Space by AMP	Space usage by AMP information for the selected table
Table, View, Macro	Rights	Access rights for the selected object
Table, View, Macro	Users	The users who have access rights to the selected object
Table	Journal	The journal table for the selected table
Table, View, Macro	Show Definition	The text that was used to create the selected object

TERADATA
Raising Intelligence

Teradata Administrator Object Menu Options

Object menu

- Select a table, view or macro from the upper grid area and the desired submenu selection.
- This menu can also be seen by right-clicking on the object.
- Information appears in lower pane area.

Name	Type	CurrentPerm	PeakPerm	SkewFact
1 TABLE2	Table	1,019,904	2,029,568	0
2 TABLE2_NUPI	Table	1,020,928	1,020,928	13

TableName	Vproc	CurrentPerm	PeakPerm
1 TABLE2_NUPI	0	95,232	95,232
2 TABLE2_NUPI	1	95,232	95,232
3 TABLE2_NUPI	2	145,920	145,920
4 TABLE2_NUPI	3	145,920	145,920
5 TABLE2_NUPI	4	123,904	123,904
6 TABLE2_NUPI	5	123,904	123,904
7 TABLE2_NUPI	6	145,408	145,408

Elapsed 00:00:00

Ready

Space by AMP - Ifact01.TABLE2_NUPI

Lower pane

Same information as provided by earlier DBC.TableSize view.

Transient Journal Space

The Transient Journal (TJ) is a table that can actually grow larger than the maximum permanent space assigned to DBC. The TJ gets its perm space from DBC, but can grow larger than DBC's limit. If the TJ grows larger than the space assigned to DBC, writes made to the Transient Journal ignore the "out of space" error message that is given when a table exceeds the maximum permanent space of a database.

Therefore, a TJ can actually use more perm space than is assigned to DBC.

For all users (including DBC for most tables), when the given amount of free perm space reaches 0, the user will get the following error:

2644 ERR AMPS NO ROOM 'No more room in data base %DBID.';

However, DBC is actually allowed to go into Negative FREE PERM for important system tables like the Transient Journal and the Recovery Journal.

TERADATA
Raising Intelligence

Teradata Administrator Transient Journal Space

The Transient Journal may use any available space in the system.

This example illustrates that the DBC.TransientJournal table may use more perm space than is allocated to DBC.

Note that the MaxPerm of DBC is 51 MB, but its CurrentPerm is 88 MB.

Name	CurrentPerm	MaxPerm	PeakPerm	MaxSpool	PeakSpool
All	0	0	0	1,962,861,036	0
console	0	50,000	0	1,962,861,034	0
Crashdumps	0	185,139,200	0	1,024,000,000	0
DBC	88,560,640	51,298,396	91,037,184	1,962,861,036	3,096,576
DBCManager	0	20,971,520	0	1,962,861,034	19,968
dbcmngr	161,792	20,971,520	161,792	1,962,861,034	0
Default	0	0	0	1,962,861,036	0
PUBLIC	0	0	0	1,962,861,036	0
QCD	2,441,728	50,000,000	2,441,728	100,000,000	0
Spool_Reserve	0	104,857,600	0	1,962,861,034	0

Ferret Utility

To maintain data integrity, the Ferret utility (File Reconfiguration Tool) enables you to display and set various disk space utilization attributes associated with the Teradata RDBMS.

When you select the Ferret utility attributes and functions, it dynamically reconfigures the data on the disks to correspond with the selections.

Depending on the functions, Ferret can operate at the vproc, table, subtable, disk, or cylinder level.

Start Ferret from the DBW connected to the Teradata database. Note that the Teradata database must be in the Logons Enabled state.

The commands within the Ferret utility that we will discuss in this module include:

- SCOPE
- SHOWSPACE
- SHOWBLOCKS

To start the Ferret utility, enter the following command in the Supervisor screen of the DBW:

START FERRET

You will be placed in the interactive partition where the Ferret utility was started.



Ferret Utility

ferret

File Help

Ferret Utility

Release 12.00.00.04 Version 12.00.00.04
Ferret

Waiting for 2 Ferret background tasks to start
All Background tasks have been started
FERRET will run in Workload Def: WD-ConsoleM
The SCOPE has been set

Ferret ==>

Status: Reading

Enter a command:

|

Function of FERRET covered in this module:

- SHOWSPACE
- SHOWBLOCKS

From Supervisor (of Database Window), enter: START FERRET

Ferret SHOWSPACE Command

The SHOWSPACE command reports the amount of disk cylinder space currently in use and the amount of cylinder space that remains available. Use SHOWSPACE to determine if disk compaction or system expansion is required.

SHOWSPACE is a command you execute from within the Ferret utility. To start the utility, enter **START FERRET** in the Supervisor window. Within the Ferret application window, enter **SHOWSPACE** (upper or lowercase). The Showspace command reports on physical disk utilization, reported as:

- Permanent space
- Spool space
- Global Temporary space
- Journal space
- Lost disk space from disk flaws
- Free disk space

The facing page shows the results of a SHOWSPACE command. Notice the command displays the average utilization per cylinder for permanent and spool space. It displays the percentage of total available cylinders as well as the number of cylinders for all types of space.

Enter an **S** for a summary report that displays only subtotals for all AMP vprocs in the system. The facing page shows an example of a Showspace summary report, and displays the following request to determine if the report is for a single AMP or for all AMPS:

The typical percentage of cylinders used for Permanent data is 28%.

Enter an **L** for a full report that displays Pdisk and Vdisk information for all AMP vprocs in the system. The full report format displays information separately for each of the Pdisks used by an AMP vproc, as well as total space utilization for the vproc.



Ferret SHOWSPACE Command

ferret

File Help

SHOWSPACE results for the Whole AMP

Proc	Num DSU	AMP →	Perm Cyls	Data Cyls	Cyls/Cyl	Wal Cyls	Depot Cyls/Cyl	Spool Cyls	Temp Cyls	Jrnl Cyls	Free Cyls										
			Avg	% of total	#Cyl/Cyl	Avg	% of total	Avg	% of total	Avg	% of total										
			Avail	per Avail		Avail	per Avail	Avail	per Avail	Avail	per Avail										
			Cyls	Cyls	#Cyl/Cyl	Cyls	#Cyl/Cyl	Cyls	#Cyl/Cyl	Cyls	#Cyl/Cyl										
	0	→	36694	79%	28%	10133	0%	15	0%	6	59%	5%	19441	0%	0%	1	0%	0%	1	67%	245941
	1		36694	79%	27%	10063	0%	15	0%	6	60%	5%	19651	0%	0%	1	0%	0%	1	68%	246431
	2		36694	78%	28%	10198	0%	15	0%	6	63%	5%	19191	0%	0%	1	0%	0%	1	67%	245541
	3		36694	78%	28%	10218	0%	15	0%	6	61%	5%	19381	0%	0%	1	0%	0%	1	67%	245151

Status: Reading

Showspace options: **showspace /s Summary listing**
showspace /l Long listing

Enter a command:

Approximately, how large (in GB) is each AMP's Vdisk?

What is the typical percentage of cylinders (per AMP) that is used for Permanent data?

Ferret SHOWBLOCKS

The Ferret utility includes a SHOWBLOCKS command that displays the data block size and/or the number of rows per data block for a defined scope.

The SHOWBLOCKS command displays the following disk space information for a defined range of data blocks and cylinders:

- Amount of disk space in use for permanent data
- Amount of disk space in use for spool data
- Amount of disk space lost as the result of disk flaws
- Amount of free disk space remaining

You can also display the average cylinder utilization for permanent and spool data cylinders.

You can display disk space and utilization information for each AMP in the defined scope. You can also display information for each Pdisk or disk storage unit (DSU).

SHOWBLOCKS /M – displays subtable information.

SHOWBLOCKS /L – displays minimum, average, and maximum number of rows per block.

SHOWBLOCKS /S – displays table level information – doesn't display empty tables (tables with no rows).

Another option to set the scope for the table (example on facing page) is to use the following command:

- SCOPE TABLE ("PD.Employee" 0)

The facing page poses this question – “How large (in MB) is primary data subtable?”

The solution is (typical block size) x (total number of blocks).

124 sectors x 512 bytes = 62 KB

62 KB x 30 blocks = 1920 KB or 2 MB.

TERADATA
Raising Intelligence

Ferret SHOWBLOCKS Command

The table id for PD.Employee is
0 1243 0 {0x0000 0x04DB 0x0000}

Ferret ==>
scope table 0 1243 0
The SCOPE has been set
Ferret ==>
showblocks /s
Showblocks has been started on all AMP vprocs in the SCOPE.
Type 'ABORT' to stop the command before completion

Table ID	Distribution of data block sizes (by range of number of sectors)																Data block size statistics (sectors)	Total Number of Blocks	Total Number of Cylinders
	1-	2-	4-	8-	16-	32-	48-	64-	80-	96-	112-	128-	160-	192-	224-	Min			
0 1243	1	3	7	15	31	47	63	79	95	111	127	159	191	223	255	77	124	127	

How large (in MB) is this primary data subtable?

Note: The Summary information display only provides size information about the primary data subtable of a table. Fallback and index subtables are not included.

Showblocks options:

- [showblocks /s](#) displays table information
- [showblocks /m](#) displays subtable information
- [showblocks /l](#) displays rows per block information

Ferret SHOWBLOCKS – Subtable Detail

The Ferret ShowBlocks utility also allows you to view block sizes down to the subtable level. The /m option provides this level of detail

Notes about Subtable IDs (shown in decimal in ShowBlocks report):

0	– Header
1024	– Primary data subtable
2048	– Fallback subtable
1028	– 1 st Secondary Index subtable
2052	– 1 st Secondary Index Fallback subtable
1032	– 2 nd Secondary Index subtable
2056	– 2 nd Secondary Index Fallback subtable
1536	– 1 st Reference Index subtable
2560	– 1 st Reference Index Fallback subtable
1792	– 1 st BLOB or CLOB subtable
2816	– 1 st BLOB or CLOB subtable
1794	– 2 nd BLOB or CLOB subtable
2818	– 2 nd BLOB or CLOB subtable

The facing page poses this question – “How large (in MB) is first secondary index?”

The 1st secondary index subtables have subtable IDs of 1028 and 2052.

The solution is (typical block size) x (total number of blocks).

114 sectors x 512 bytes = 57 KB

57 KB x 16 blocks x 2 = 1824 KB or 1.8 MB.

Note: Subtable ID of 1028 is 16 blocks and the fallback (2052) is also 16 blocks. The 1st secondary index uses a total of 32 blocks.

TERADATA
Raising Intelligence

Ferret SHOWBLOCKS – Subtable Detail

ferret

File Help

showblocks /m

Showblocks has been started on all AMP vprocs in the SCOPE.
Type 'ABORT' to stop the command before completion

Table ID	Distribution of data block sizes (by range of number of sectors)												Data block size statistics			Total Number of	Total Number of			
	1-	2-	4-	8-	16-	32-	48-	64-	80-	96-	112-	128-	160-	192-	224-	Min	Avg	Max	Data Blocks	Cylinders
0 1243 PD.Employee																				
0 100%																3	3	3	2	2
1024									7%							77	124	127	30	3
1028																26	114	127	16	2
1032										33%						43	100	128	6	2
1036																71	109	128	6	2
2048											7%					77	124	127	30	3
2052																26	114	127	16	2
2056																42	100	128	6	2

Status: Reading

```
tableid 'PD.Employee'
scope table 0 1243 0
showblocks /s
showblocks /m
```

showblocks /m – This example displays the distribution of a specific table and its indexes.

Enter a command:

How large (in KB) is the 1st secondary index?

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. True or False. Space limits are enforced at the table level.
2. True or False. When you use the GIVE statement to transfer a database/user, only the tables allocated to the original database/user are transferred to the new database/user.
3. True or False. You should reserve anywhere from 20 - 25% of total available space for spool.
4. The DBC._____ view provides disk space usage at the table level and excludes table ALL.
5. The DBC._____ view only provides disk space usage at the database level.

Teradata Training

Notes

Module 44



Users, Accounts, and Accounting

After completing this module, you will be able to:

- Use Teradata accounting features to determine resource usage by user or account.
- Explain how the database administrator uses system accounting to support administrative functions.
- Use system views to access system accounting information.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Creating New Users & Databases	44-4
CREATE DATABASE Statement	44-6
CREATE USER Statement	44-8
CREATE USER and the Data Dictionary	44-10
CREATE USER and the Data Dictionary (cont.)	44-12
MODIFY USER Statement	44-14
Teradata Administrator – Tools Menu > Create Options	44-16
Creating and Using Account IDs	44-18
Using Account IDs with Logon	44-18
Dynamically Changing an Account ID	44-20
Syntax	44-20
Examples	44-20
Account Priorities	44-22
Account String Expansion	44-24
ASE Variables:	44-24
ASE Accounting Example	44-26
Background Information	44-26
Tasks	44-26
System Accounting Views	44-28
DBC.AccountInfo[x]	44-28
DBC.AMPUsage	44-28
Dictionary Tables Accessed	44-28
DBC.AccountInfo[X] View	44-30
Example	44-30
DBC.AMPUsage View	44-32
DBC.AMPUsage View — Examples	44-34
Example	44-34
Users, Accounts & Accounting Summary	44-36
Review Questions	44-38
Lab Exercise 44-1	44-40

Creating New Users & Databases

As the database administrator, you create system databases and tables and assign user privileges and access rights to tables.

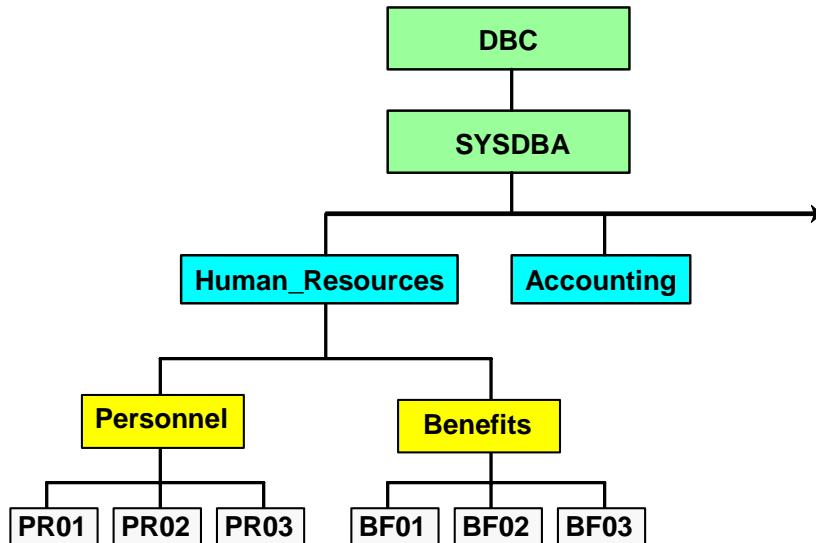
To perform the above tasks, you must:

- Determine database information content and create macros to ensure the referential integrity of the database.
- Define authorization checks and validation procedures.
- Perform audit checks on the database for LOGON, GRANT, REVOKE and other privilege statements.

You can give the authority to use the CREATE DATABASE or CREATE USER statements to any application user. He or she may then create other system users or databases from his or her own space, or if specifically authorized, from the space of another system database or user.



Creating New Users and Databases



You can grant CREATE DATABASE authority to any user.

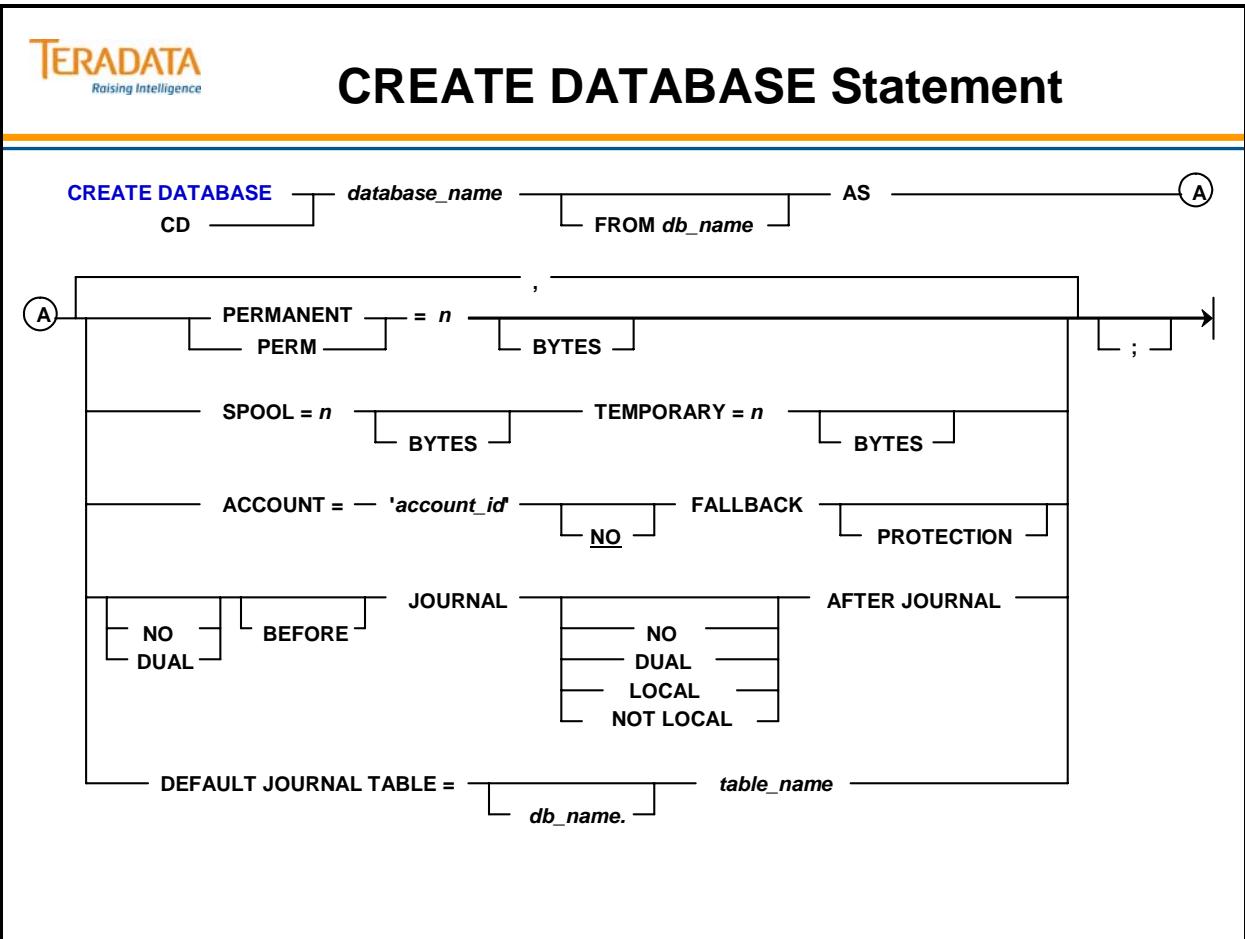
The user may then create other users and databases from:

- The user's own space, or
- The space of another user or database (if authorized).

CREATE DATABASE Statement

As the database administrator, you use the CREATE DATABASE statement to add new databases to the existing system. The permanent space for new databases you create comes from the immediate parent database or user. A database becomes a uniquely named collection of tables, views, macros, triggers, stored procedures, and access rights.

The spool definition is not relevant to a database. It establishes the default value for objects you create within the database hierarchy.



CREATE USER Statement

The CREATE USER statement enables you to add new users to the system. The permanent space for these new users comes from the immediate parent database or user.

Users have passwords while databases do not. User passwords allow users to log on to the Teradata database and establish sessions.

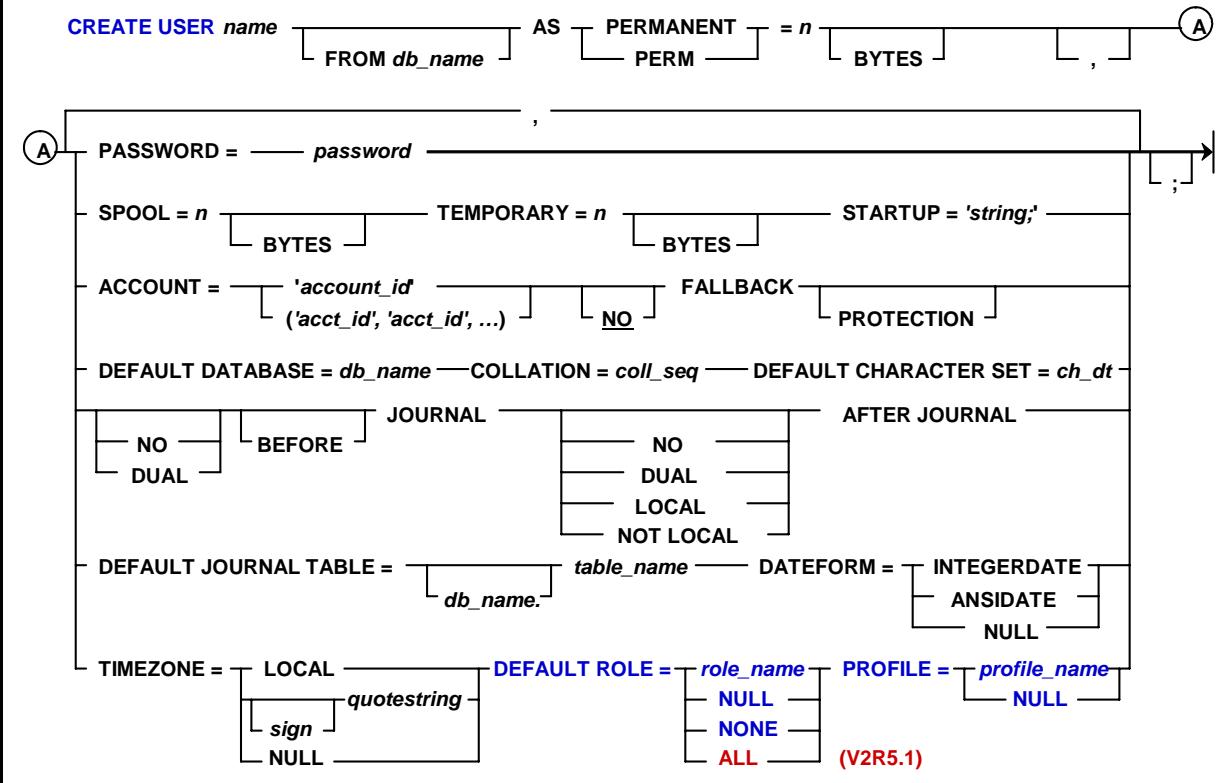
When you create a new user, you also create a temporary password for the user. When the user logs on for the first time, he or she is prompted to change the password. Note: This assumes the password expiration time period is not set to 0.

If a user forgets the password, you can assign a new temporary password. (As another option, you can set user passwords not to expire.)

Acronym: ch_dt – Character Data Type



CREATE USER Statement



CREATE USER and the Data Dictionary

In addition to creating a new system user, the CREATE USER statement also defines space.

A user is associated with a password and an account, and can log on, establish a session, and execute SQL statements. A user, not a database, performs these actions.

Notice the entries the CREATE USER statement makes in the data dictionary.
Default values associated with the CREATE USER statement are:

<u>Entry</u>	<u>Defaults to the value of</u>
FROM database	Current CREATOR
SPOOL	Same value as the OWNER
TEMPORARY	Same value as the OWNER
STARTUP	Null (no startup string)
ACCOUNT	Immediate OWNER'S first account ID
DEFAULT DATABASE	Username

There are also two types of rights granted automatically when you use the CREATE USER statement:

- The rights granted to a newly created user or database on itself.
- The rights granted on a newly created user, database, or object to the creating user.

By issuing a CREATE USER statement, the creator gains certain automatic rights over the created object.

As shown on the facing page, the database administrator logged on as Sysdba and creates tfact06:

```
CREATE USER tfact06 AS PERM = 10e6, SPOOL = 100e6,
  PASSWORD = secure1time;
```



CREATE USER and the Data Dictionary

EXPLAIN

CREATE USER tfact06 AS PERM = 10e6, SPOOL = 100e6, PASSWORD = secure1time;

Explanation

- 1) First, we lock data base tfact06 for exclusive use.
- 2) Next, we lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.DataBaseSpace.
- 3) We lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.AccessRights.
- 4) We lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.Parents.
- 5) We lock a distinct DBC."pseudo table" for write on a RowHash to prevent global deadlock for DBC.Owners.
- 6) We lock DBC.DataBaseSpace for write, we lock DBC.AccessRights for write, we lock DBC.Parents for write, we lock DBC.Owners for write, we lock DBC.Accounts for write on a RowHash, we lock DBC.DBase for write on a RowHash, and we lock DBC.DBase for write on a RowHash.
- 7) 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 with no residual conditions.
 - 2) We do a single-AMP ABORT test from DBC.Roles by way of the unique primary index with no residual conditions.
 - 3) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
 - 4) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
 - 5) We do an INSERT into DBC.DBase.
 - 6) We do a single-AMP UPDATE from DBC.DBase by way of the unique primary index with no residual conditions.
 - 7) We do a single-AMP RETRIEVE step from DBC.Parents by way of the primary index with no residual conditions into Spool 1 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 1 by row hash.
- 8) We do an all-AMPs MERGE into DBC.Owners from Spool 1 (Last Use).

CREATE USER and the Data Dictionary (cont.)

Several steps are performed in parallel during the CREATE USER statement.



CREATE USER and the Data Dictionary (cont.)

- 9) We execute the following steps in parallel.
 - 1) We do an INSERT into DBC.Owners.
 - 2) We do a single-AMP RETRIEVE step from DBC.Parents by way of the primary index with no residual conditions into Spool 2 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 2 by row hash.
- 10) We do an all-AMPs MERGE into DBC.Parents from Spool 2 (Last Use).
- 11) We execute the following steps in parallel.
 - 1) We do an INSERT into DBC.Parents.
 - 2) We do an INSERT into DBC.Accounts.
 - 3) We do a single-AMP RETRIEVE step from DBC.AccessRights by way of the primary index into Spool 3 (all_amps), which is redistributed by hash code to all AMPs.
- 12) We execute the following steps in parallel.
 - 1) We do a single-AMP RETRIEVE step from DBC.AccessRights by way of the primary index into Spool 3 (all_amps), which is redistributed by hash code to all AMPs.
 - 2) We do an all-AMPs RETRIEVE step from DBC.AccessRights by way of an all-rows scan into Spool 4 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 4 by row hash.
- 13) We do an all-AMPs JOIN step from DBC.Owners by way of a RowHash match scan, which is joined to Spool 4 (Last Use). DBC.Owners and Spool 4 are joined using a merge join. The result goes into Spool 3 (all_amps), which is redistributed by hash code to all AMPs. Then we do a SORT to order Spool 3 by row hash.
- 14) We do an all-AMPs MERGE into DBC.AccessRights from Spool 3 (Last Use).
- 15) We flush the DISKSPACE and AMPUSAGE caches.
- 16) We do an all-AMPs ABORT test from DBC.DataBaseSpace by way of the unique primary index.
- 17) We do an INSERT into DBC.DataBaseSpace.
- 18) We do an all-AMPs UPDATE from DBC.DataBaseSpace by way of the unique primary index with no residual conditions.
- 19) We flush the DISKSPACE and AMPUSAGE caches.
- 20) We spoil the parser's dictionary cache for the database.
- 21) 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.

MODIFY USER Statement

The MODIFY USER statement enables you to change the options of an existing user.

Options you can change without the DROP DATABASE privilege include:

- Password
- Startup string
- Default database
- Collation
- Fallback Protection default
- Default Dateform
- Default Character Set data type
- Timezone
- Permanent journal default options

Options requiring the DROP DATABASE privilege are:

- PERMANENT space limit
- SPOOL space limit
- TEMPORARY space limit
- Account codes
- Release password lock
- DROP DEFAULT JOURNAL TABLE
- Role
- Profile

The FOR USER option effectively established a temporary password that can be used to logon one time by the user. This option is only effective if the ExpirePassword attribute (set in DBC.SysSsecDefaults or a profile) is set to a value greater than 0.

MODIFY USER RobertSmith AS PASSWORD = secret FOR USER;

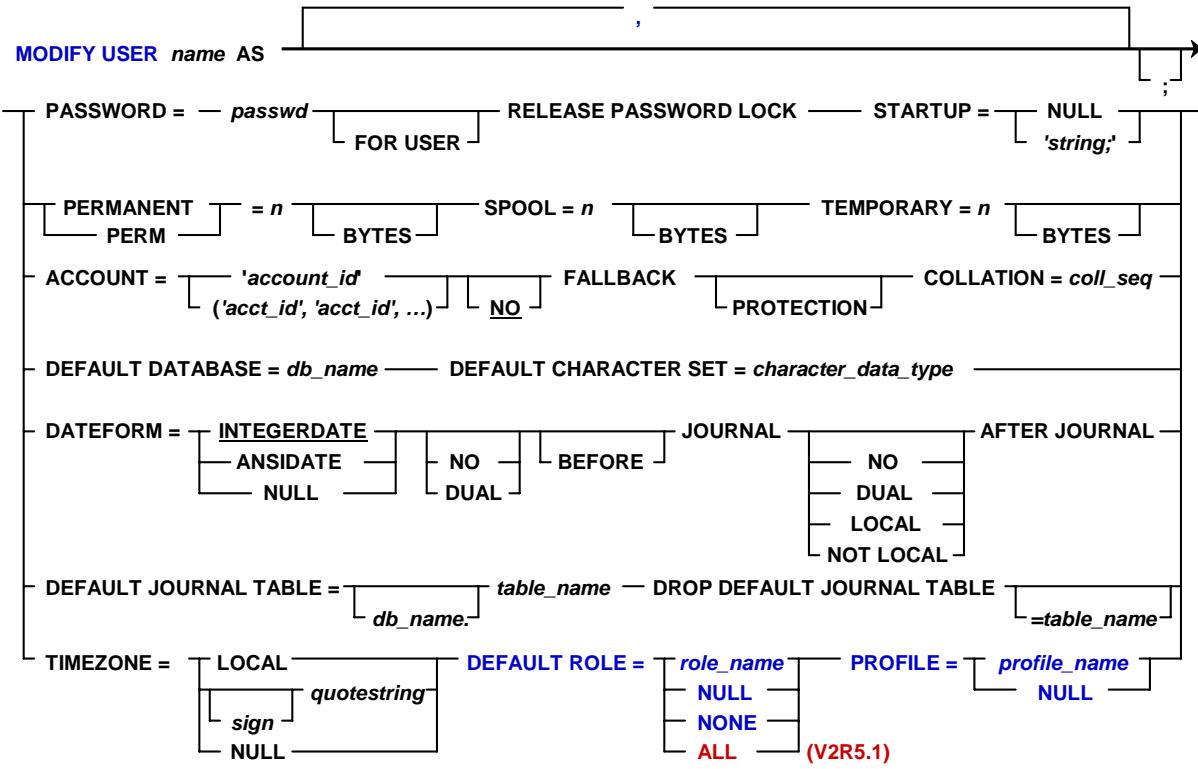
The existing password immediately expires and is replaced by “secret”. In this example, “secret” is effectively a temporary password that allows a one-time logon. The value for PasswordChgDate is reset to 0.

Note: The PasswordChgDate column is also set to 0 when a new user is created – assuming that ExpirePassword is set to a value greater than 0.

The temporary password expires immediately when the user logs on for the first time and the user needs to select a new, permanent password at that time. Another option is to use the MODIFY USER command without the FOR USER option.



MODIFY USER Statement



Teradata Administrator – Tools Menu > Create Options

The Tools menu provides the following options.

Menu Selection	Function / Options
Create	Create an entirely new object – Database, Table, User, Profile, or Role.
Grant/Revoke	Grant or revoke general access privileges to users. Options include Object Rights, System Rights, Logon Rights, or Column Rights.
Administer Profiles	Create and manage Profiles for users. (V2R5 feature)
Administer Roles	Create and manage Roles. (V2R5 feature)
Clone User	Create a new user either identical or closely related to an existing user.
Modify User	Change the specifications of an existing user.
Access Logging	Create and manage Access Log rules.
Query Logging	Create and manager Query Log rules. (V2R5 feature)
Move Space	Reallocate permanent disk space from one database to another (efficient if not a direct descendent or parent).
Query	Create, modify, test, or run SQL query scripts.
Options	Configure the operational preferences for Teradata Administrator.

TERADATA
Raising Intelligence

Teradata Administrator Tools Menu > Create Options

Teradata Administrator can be used to create and manage users and databases.

Tools menu

- Selections to create and modify databases and users, grant/revoke access rights, and send ad hoc query requests to Teradata.
- Options include the ability to clone a user, move space, and set preferences.
- This example illustrates how to create a database by completing the entries.

Name	CommentString
F	1 AU

Creating and Using Account IDs

When you create a user, you can specify one or more account IDs that a new user can specify. Account codes may be used to track system CPU, I/O usage, or space usage. When the user logs on, the user can specify a valid account ID, or let the first account ID in the user row (from CREATE or MODIFY USER) become the default.

You should determine an account ID scheme for ease of accounting and priorities.

Account IDs may begin with the characters \$L, \$M, \$H, or \$R to identify the priorities low, medium, high, and rush, respectively. The relative level of CPU service is 1, 2, 4, and 8, respectively. These priority levels will be discussed on the following pages.

Using Account IDs with Logon

All logons require an account ID. A user can submit an explicit account ID by including it in the logon string. It must be a valid ID specified in the last CREATE or MODIFY USER statement. If no ID is specified in the CREATE or MODIFY statements, it defaults to the ID of the immediate owner's database.

Note:

batch logon syntax:

.LOGON tdpid/user_name, password, 'account_ID';

BTEQ Interactive logon syntax:

.LOGON tdpid/user_name,, 'account_ID' **(note the two commas)**
Enter password when prompted



Creating and Using Account IDs

<code>CREATE USER tfact07 FROM Sysdba AS PERM = 10e6 ,SPOOL = 100e6 ,PASSWORD = secure12 ,FALLBACK ,ACCOUNT = ('\$M', '\$M_9038', '\$M_9038_&S&D&H', '\$H_9038');</code>	Names user Name of immediate owner in hierarchy Amount of Permanent space Maximum amount of Spool space Initial password Default protection type when creating a table Default account code – medium priority Optional account code – medium priority Optional account code – medium priority with ASE Optional account code – high priority
--	---

Logon - all logons require an account ID. The first account ID is the default account ID.

batch logon syntax: `.LOGON tdpid/username,password,'account_id';`

Example:	<code>.LOGON educ/tfact07,secure12,'\$H_9038'</code>
-----------------	--

BTEQ Interactive logon syntax: `.LOGON tdpid/username,,'account_id';`

Example:	<code>.LOGON educ/tfact07,'\$H_9038'</code>
-----------------	---

Enter password when prompted	<code>*****</code>
-------------------------------------	--------------------

Dynamically Changing an Account ID

You can dynamically change your Account ID without logging off and logging back on. One reason you may want to do this is to change your session's priority. This is also called "nicing a query". "Nicing" is a UNIX term that means manipulating the scheduling priority of a "running" task. You typically "nice" a query to re-prioritize jobs. For instance, you could nice a query to a higher priority to run a business-critical job sooner than under its originally defined priority.

Self-nicing refers to a user specifying changes on his/her own request or session.

Asynchronous nicing refers to a super user or system administrator manipulating another user's account.

Use the SET SESSION ACCOUNT statement to change your performance group (account priority) for the next SQL query you run, or for all jobs for the remainder of the current session.

Syntax

For the next SQL statement:

SET SESSION ACCOUNT = '*Account_ID*' FOR REQUEST;

For the remainder of the current session:

SET SESSION ACCOUNT = '*Account_ID*' FOR SESSION;

Examples

You cannot change a priority to exceed the priority originally defined by the performance group for an account or to a forbidden priority level. The following chart shows three accounts and the defined, permitted and forbidden priorities.

<i>Account</i>	<i>Defined Priority</i>	<i>Priority Definition</i>	<i>Permitted Priority Changes</i>	<i>Forbidden Priority Changes</i>
Sales	\$H	High	<=\$H	\$R
Marketing	\$M	Medium	<\$H	\$H
Development	\$L	Low	None	>\$L

You can see that you cannot change marketing's account priority to high, because it exceeds the group's original priority definition and is a forbidden priority for the account.

As another example, you can change sale's priority group to low or medium, because they do not exceed the groups' original priority definition and are not forbidden for the account.

Lastly, you cannot change development's priority. The chart shows that no priority changes are permitted and that the account cannot have any priority that exceeds low.



Dynamically Changing an Account ID

- You can change your Account ID without logging off. This may be done to re-prioritize a query. This is also referred to as “nicing a query”.
- You can change Account IDs for the next SQL statement you run, or for all jobs for the remainder of the current session.
- To change Account IDs, use the SET SESSION ACCOUNT statement:

Syntax:

For the next SQL statement : `SET SESSION ACCOUNT = 'Account_ID' FOR REQUEST;`

For the rest of the current session: `SET SESSION ACCOUNT = 'Account_ID' FOR SESSION;`

Example:

For the rest of the tfact07 session: `SET SESSION ACCOUNT = '$H_9038' FOR SESSION;`

- Note: You can only use valid account IDs. Therefore, you cannot exceed the priority defined by the performance groups in your account ID.

Account Priorities

Account IDs may begin with the characters \$L, \$M, \$H, or \$R to identify the priorities low, medium, high, and rush, respectively. The relative level of CPU service is 1, 2, 4, and 8, respectively.

The Priority Scheduler facility lets you use codes to assign users to performance groups using these levels and user-defined levels of CPU usage. The Priority Scheduler facility will be described in detail later in the course.

You can design billing algorithms to reflect the usage of higher or lower account priorities. That way, a user with \$H account priority is charged more for using system resources than a user with \$L account priority.

Character	Priority	CPU Service	Comments
\$L	Low	1	Consider using for queries not needed immediately, such as batch jobs.
\$M	Medium	2	Consider using for complex ad hoc queries.
\$H	High	4	Consider using for tactical or OLTP queries.
\$R	Rush	8	Use for very critical queries

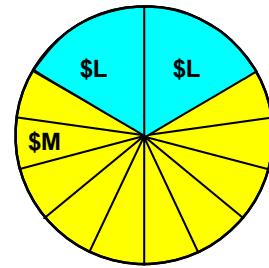
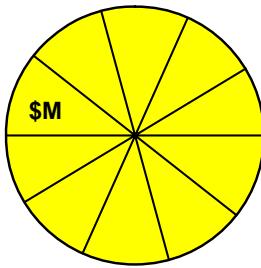


Account Priorities

<u>Performance Groups</u>	<u>Default Weights</u>	<u>Possible Uses</u>
\$L	5	Low – consider using for low priority queries (e.g., batch jobs)
\$M	10	Medium – consider using for complex ad hoc queries
\$H	20	High – consider using for index access queries (e.g., tactical)
\$R	40	Rush – consider for critical index access user queries

Performance Group	Active Sessions	Relative Weight Calc.	Percent Allocated
\$L	N	0	0
\$M	Y (10)	10/10	100
\$H	N	0	0
\$R	N	0	0

Performance Group	Active Sessions	Relative Weight Calc.	Percent Allocated
\$L	Y (2)	5/15	33.3
\$M	Y (10)	10/15	66.7
\$H	N	0	0
\$R	N	0	0



Account String Expansion

Account String Expansion (ASE) is an optional feature that enables you to use substitution variables in the account ID portion of the user's logon string. These variables enable you to include date and time information in the string. You must explicitly modify a user's logon to order to use ASE. The variables are resolved at logon time or at actual SQL execution time.

Account strings cannot exceed 30 characters. If, as a result of string expansion, you generate a string longer than 30 characters, the system truncates all characters to the right of position 30. Separation characters, such as colons in time fields and slashes in dates, are included in the character count.

ASE Variables:

- &L The **logon time stamp** variable causes the logon time stamp to be inserted into the account string. The full logon time stamp consists of 15 characters and becomes truncated if &L is placed in position 17 or higher. The value inserted into AMPUsage is established at logon time. It does not change unless the user logs off then logs on again.
- &D The **date** variable causes the date to be inserted into the account string. The value becomes truncated if you place &D at, or to the right of, position 26 or higher. You can use truncation to monitor resources on a yearly or monthly basis.
- &T The **time** variable inserts the time of day into the account string. The value becomes truncated if you place &T at, or to the right of, position 26 or higher. You can use truncation to monitor resources hourly or by the minute. This variable allows for one-second granularity, thus causing a row to be written for virtually every individual SQL request.
- &H The **hour** variable inserts the hour of the day into the account string. The inserted value consists of two characters and becomes truncated if you place &H to the right of position 30.
- &I The **logon host ID/session number/request number** variable inserts the logon host ID, the session number and the request number into the account string.
- &S The **session number** variable inserts the current session number into the account string.



Account String Expansion

- ASE is a mechanism to provide more detailed utilization reports and user accounting data.
- You may add the following substitution variables to a user's account string. The system resolves the variables at logon or at SQL statement execution time.

&D	Date	(YYMMDD)
&H	Hour	(HH)
&T	Time	(HHMMSS)
&L	Logon timestamp	(YYMMDDHHMMSS.hh)
&I	Logon hostid, session number, request number	(LLLL SSSSSSSSS RRR RRR RRR)
&S	Session number	(SSSSSSSS)
- ASE increases AMPUsage granularity.
 - Consider using &S&D&H (session & day & hour) - accumulate for each unique account, session number, day and hour.
 - Note for queries that span multiple hours, the time will be accumulated in its entirety to the query's start hour.

ASE Accounting Example

Background Information

Two existing users, TFACT01 and TFACT02, logged onto the system using an account string defined as &S&D&H. The DBC.Acctg table contains a number of rows generated by the ASE feature.

Logon example: **.LOGON DBC/tfact01, password, '\$M_9038_&S&D&H';**

Tasks

You need to create a table, view, and a number of reports that provide billing and resource usage information based on the statistics collected by the AMPUsage view.

Step 1. Create AmpUsageSum table.

Create a table to hold the collected statistics from the AMPUsage view. This table will serve as the basis for all of the other objects that you create. This is a history table since it contains stored historical data.

Step 2. Populate AMPUsageSum table.

After you build the AmpUsageSum table, use the INSERT command to populate it with row information from the DBC.AMPUsage view.

Step 3. Create Usage view

Use the CREATE statement to combine columns from the DBC.AMPUsage view and DBC.LogOnOff view into the Usage view.

Step 4. Create billing and resource usage reports.

Once the view is completed, construct SQL statements to SELECT information from the Usage view to create billing and resource usage reports.



ASE Accounting Example

- Users TFACT01 and TFACT02 each log on with **\$M_9038_&S_&D&H** account string.
 - Each hour a new row is placed in Acctg.
- Impact of using ASE variables with Acctg.

ASE Variable	Performance Impact	Data Capacity Impact
none	Negligible	1 row per account per AMP.
&D	Negligible	1 row per account per day per AMP.
&H	Negligible	1 row per account per hour per AMP (all days go into 1 hour).
&D&H	Negligible	1 row per account per hour per day per AMP.
&S&D&H	Negligible	1 row per account per session per hour per day per AMP.
&L	Negligible	1 row per logon (LAN) or session pool.
&T	Potentially Non-negligible	1 row per query per AMP.

- Perform the following tasks to extract accounting information:
 - Step 1. Create AMPUsageSum table.
 - Step 2. Populate AMPUsageSum table.
 - Step 3. Create Usage view.
 - Step 4. Create billing and resource usage reports.

System Accounting Views

The Teradata RDBMS provides two system-supplied views to support accounting functions.

DBC.AccountInfo provides information about valid accounts, and DBC.AMPUsage provides information about the usage of each AMP vproc by user and account.

DBC.AccountInfo[x]

The DBC.AccountInfo[x] view provides information about valid accounts for a specific user. The information provided is based on data from the DBC.Accounts table in the data dictionary. Each time a CREATE or MODIFY statement indicates an account ID, a row is either inserted or updated in the DBC.Accounts table.

(When you use restricted views, you must be the requester or have modify rights turned on.)

DBC.AMPUsage

The DBC.AMPUsage[X] views provide information about the usage of each AMP vproc for each user and account. It is based on information in the DBC.Acctg table in the data dictionary and supplies information about AMP CPU time consumed, and the number of AMP to DSU read and write operations generated by a given user or account. It also tracks the activities of any console utilities.

Each time a user logs on or submits an SQL request; a row is either inserted or updated in the DBC.Acctg table. If the user_name/account_name does not exist, then a new row is inserted. If the row already exists in the DBC.Acctg table, then it is updated. The rows in this table track how much AMP usage the specific user_name/account_name generates. This information may be used to bill an account for system resource use.

Dictionary Tables Accessed

DBC.Accounts
DBC.Acctg



System Accounting Views

View

Description

DBC.AccountInfo[X]

Returns each Account Name (Account ID) associated with a user (for users the requestor owns).

DBC.AMPUsage[X]

Provides information about I/O and AMP CPU usage by user and account.

DBC.AccountInfo[X] View

The DBC.AccountInfo[X] views shown on the facing page provide information about each user and the valid account codes associated with each user. When the requesting user indicates the [X] view, they can only see information about users that they own or have modify rights on.

The UserOrProfile column is new with V2R5 and indicates whether the user is an actual user or a profile.

Example

The SQL statement on the facing page requests a list of all users with a valid HIGH priority account code.



DBC.AccountInfo[X] View

Provides information about each user and the valid account codes associated with each.

(AccountInfoX - shows users and accounts the requestor owns or has modify rights to).

DBC.AccountInfo[X]

UserName	AccountName	UserOrProfile (V2R5)
----------	-------------	----------------------

Example:

Identify all users with a valid HIGH priority code.

```
SELECT      *
FROM        DBC.AccountInfo
WHERE       AccountName LIKE '$H%'
ORDER BY    1 ;
```

Example Results:

UserName	AccountName	UserOrProfile
AU	\$H_9038	User
Cust_Service_Gold	\$H_&S_&D&H	Profile
Employee	\$H_&S_&D&H	Profile
Students	\$H_9038	User
Sysdba	\$H_9038	User
TDPUSER	\$H	User
tfact01	\$H_9038_&S_&D&H	User
tfact07	\$H_9038	User

DBC.AMPUsage View

The DBC.AmpUsage[X] views use the underlying DBC.Acctg table to provide accounting information by username and account. You can update this view. This view provides CPU activity and logical I/O counts explicitly requested by the following two sources:

- AMP database software
- File system that is running in the context of an AMP worker task

This view can be used to determine which user or users are consuming CPU and I/O resources on a system.

The system requests I/Os to execute a step in the user's query. The DBC.AmpUsage view does not include I/Os the operating system performs for swapping or I/Os caused by parsing the user's query. The system charges a logical I/O even if the segment you request is cached and no physical I/O is done.

Column definitions in this view include:

<u>Column</u>	<u>Definition</u>
Vproc	The virtual processor ID
VprocType	AMP
Model	System model (e.g., 5500, etc.)



DBC.AMPUsage View

AMPUsage is an updateable view that uses the DBC.Acctg table to provide accounting information by username and account.

This view can be used to determine which users are consuming CPU and/or I/O resources.

AMPUsage will accumulate CPU and I/O usage for every unique account.

DBC.AMPUsage[X]

AccountName	UserName	CPUTime	DiskIO
Vproc	VprocType	Model	

CPUTime: Total number of AMP CPU seconds used (increments of 1/100 second).

DiskIO: Total number of logical disk I/O operations.

Vproc: AMP Vproc number

VprocType: AMP

Model: Model number (e.g., 5500)

DBC.AMPUsage View — Examples

Example

The SQL statement on the facing page requests totals for CPU time and I/O for user TFACT03. The totals are aggregates of all resources used across all AMP vprocs. The result returns six rows, one for each unique account ID that has been expanded.



DBC.AMPUsage View — Examples

Example:
Show CPU time
and I/O totals for
a single user.

```
SELECT      UserName      (CHAR(10))
           ,AccountName   (CHAR(25))
           ,SUM (CPUTime)  (FORMAT 'zzzz.99')
           ,SUM (DiskIO)   (FORMAT 'zzz,zzz,999')
           FROM        DBC.AMPUsage
           WHERE       UserName = 'tfact03'
           GROUP BY    1, 2
           ORDER BY    3 DESC ;
```

Example Results:

Note:
The Account IDs for
TFACT03 are:
\$M_9038
\$M_9038_&S_&D&H
\$L_9038_&D&H

UserName	AccountName	Sum(CPUTime)	Sum(DiskIO)
TFACT03	\$M_9038	37,259.45	62,339,216
TFACT03	\$M_9038_000001004_07031617	924.32	1,821,581
TFACT03	\$M_9038_000001004_07031618	389.25	710,619
TFACT03	\$M_9038_000001005_07031617	184.63	391,912
TFACT03	\$L_9038_07031908	113.28	119,457
TFACT03	\$L_9038_07031909	12.56	20,115

To reset counters for ALL rows or selected rows., you can use the DBC.ClearAccounting macro.

```
SHOW MACRO DBC.ClearAccounting;
REPLACE MACRO DBC.ClearAccounting
AS ( UPDATE Acctg SET CPU = 0, IO = 0 ALL; );
```

Users, Accounts & Accounting Summary

The facing page summarizes some important concepts regarding this module.



Users, Accounts & Accounting Summary

- To establish execution time priorities, use the first two characters in the account code and the performance groups.
- Your position in the hierarchy does not affect your priority.
- You can define accounting mechanisms:
 - Charge-back billing
 - System usage reporting
 - Capacity planning
 - Performance analysis
- To reset data dictionary tables used to collect accounting information, use:
 - DBC.ClearAccounting macro
 - DBC.ClearPeakDisk macro

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. **True or False.** You can only give the authority to use the CREATE DATABASE and CREATE USER statements to certain types of users.
2. **True or False.** An individual user with a \$L priority will always receive less CPU time than a user with a \$M priority.
3. **True or False.** A user can use the MODIFY USER statement to change their password, default database, and date format.
4. When creating a new user, which of the following option(s) default to the immediate owner's value.

 - A. SPOOL
 - B. FALBACK PROTECTION
 - C. All of the Account_IDs
 - D. DEFAULT DATABASE
5. When creating a new user, which of the following option(s) are required with the CREATE USER command. _____
 - A. SPOOL
 - B. PERMANENT
 - C. User name
 - D. PASSWORD

Lab Exercise 44-1

The following pages describe the tasks for this lab exercise.



Lab Exercises

Lab Exercise 44-1

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to view information in the data dictionary regarding space usage and accounting information (use Appendix C).

Tasks

1. Using the DBC.DiskSpace view, find the total disk storage capacity of the system on which you are logged on:

Total capacity _____

2. Using the same view, find how much of the space is currently in use:

Current space utilization _____

Write a query to show what percentage of system capacity is currently in use. _____ %

OPTIONAL: Write a query to show which databases/users are currently using (current perm) the largest percentage of their max perm space limit (group by database/user).

Lab Exercise 44-1 (cont.)

The following pages describe the tasks for this lab exercise.



Lab Exercises

Lab Exercise 44-1 (cont.)

Tasks

3. Using the DBC.Databases view, find the total number of databases and users defined in the system.

Total row count (databases and users) _____

Using this view, how many users are there? _____

Using this view, how many databases are there ? _____

Who is the creator of AP? _____

Who is the owner of AP? _____

4. Using the TableSize view, find the name and size of each table in the DBC user. List the Data Dictionary tables in DESCending order by size.

List the six largest tables:

1. _____ 2. _____ 3. _____

4. _____ 5. _____ 6. _____

Lab Exercise 44-1 (cont.)

The following pages describe the tasks for this lab exercise.



Lab Exercises

Lab Exercise 44-1 (cont.)

Tasks

5. Using the DBC.AMPUsage view, find the number of AMP vprocs defined on your system.
(HINT: Use a WHERE condition to reduce the number of DD/D table rows considered.)

Number of AMPS _____

6. Using the DBC.AccountInfo view, list all of your valid account codes.

7. Using the DBC.AMPUsage view, write a query to show the number of AMP CPU seconds and logical disk I/Os that have been charged to your:

User ID _____ Seconds _____ I/Os _____

Teradata Training

Notes

Module 45



Access Rights

After completing this module, you will be able to:

- Use the DBC.AllRights, DBC.UserRights and DBC.UserGrantedRights views to obtain information about current users.
- Use views and macros to access information about privileges.
- Use the GRANT and REVOKE statements to assign and remove access rights.
- Understand the impact of the GIVE statement with access rights.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Privileges/Access Rights.....	45-4
Access Rights Mechanisms.....	45-6
Automatic Rights	45-6
Explicit Rights.....	45-6
Ownership Rights.....	45-6
Access Rights Views.....	45-6
CREATE TABLE – Automatic Rights	45-8
CREATE USER – Automatic Rights.....	45-10
Example	45-10
Implicit, Automatic, and Explicit Rights	45-12
Example	45-12
GRANT Command	45-14
Granting Rights at Database Level	45-16
GRANT Rights at the Table or Column Level	45-18
Access Rights and Triggers.....	45-18
REVOKE Command.....	45-20
REVOKE Recipients.....	45-20
Revoking Non-Existent Rights	45-22
Example	45-22
Removing a Level in the Hierarchy	45-24
Transfer Ownership.....	45-24
Delete User or Database.....	45-24
Drop User.....	45-24
Access Rights	45-24
Inheriting Access Rights	45-26
Example	45-26
The GIVE Statement and Access Rights	45-28
Example	45-28
Access Rights and Views	45-30
Access Rights and Nested Views.....	45-32
System Views for Access Rights	45-34
DBC.AllRights[V][X].....	45-34
DBC.UserRights[V].....	45-34
DBC.UserGrantedRights[V]	45-34
DBC.AllRights[X] and DBC.UserRights Views	45-36
DBC.UserGrantedRights View	45-38
Teradata Administrator – Grant/Revoke Rights	45-40
Teradata Administrator – Rights on DB/User.....	45-42
Access Rights Summary	45-44
Review Questions	45-46

Privileges/Access Rights

Your privileges or access rights define the types of activities you can perform during a session.

The following operations require that you have specific privileges:

- CREATE
 - DROP
 - REFERENCES
 - INDEX
 - SELECT
 - UPDATE
 - INSERT
 - DELETE
 - EXECUTE
 - EXECUTE PROCEDURE
 - CHECKPOINT
 - DUMP
 - RESTORE
- }

DDL

}

DML

}

Archive/Recovery

Access rights are associated with:

Users	Macros	Columns of tables
Databases	Triggers	Columns of views
Tables	Stored Procedures	
Views	User-defined Functions	

Notes:

- To use UPDATE or DELETE commands, you must have the SELECT right on the object.
- Additional rights you need to control access to performance monitoring functions are discussed in another module.
- A column can only be specified with the UPDATE or REFERENCES access right.



Privileges/Access Rights

A privilege (or access right) is the right of a specific user to perform a specified operation.

Note: Some access rights don't directly correspond to an SQL statement.



On a specified Object



Access Rights Mechanisms

The data dictionary includes a system table called DBC.AccessRights that contains information about the access rights assigned to existing users.

The DBC.AccessRights table internally has the following indexes:

- PI – NUPI (UserId, DatabaseId)
- SI – NUSI (TVMId)

Access rights may be categorized in one three ways:

- Automatic (or Default) Access Rights
- Explicit Access Rights
- Implicit (or Ownership) Access Rights

Automatic Rights

Automatic rights are privileges given to creators and, in the case of users and databases, their created objects. When a user submits a CREATE statement, new rows are inserted in the DBC.AccessRights table. All rights are automatically removed for an object when it is dropped.

Explicit Rights

Explicit rights are privileges conferred by using a GRANT statement. This statement inserts new rows into the DBC.AccessRights table. Explicit rights can be removed using the REVOKE statement.

Ownership Rights

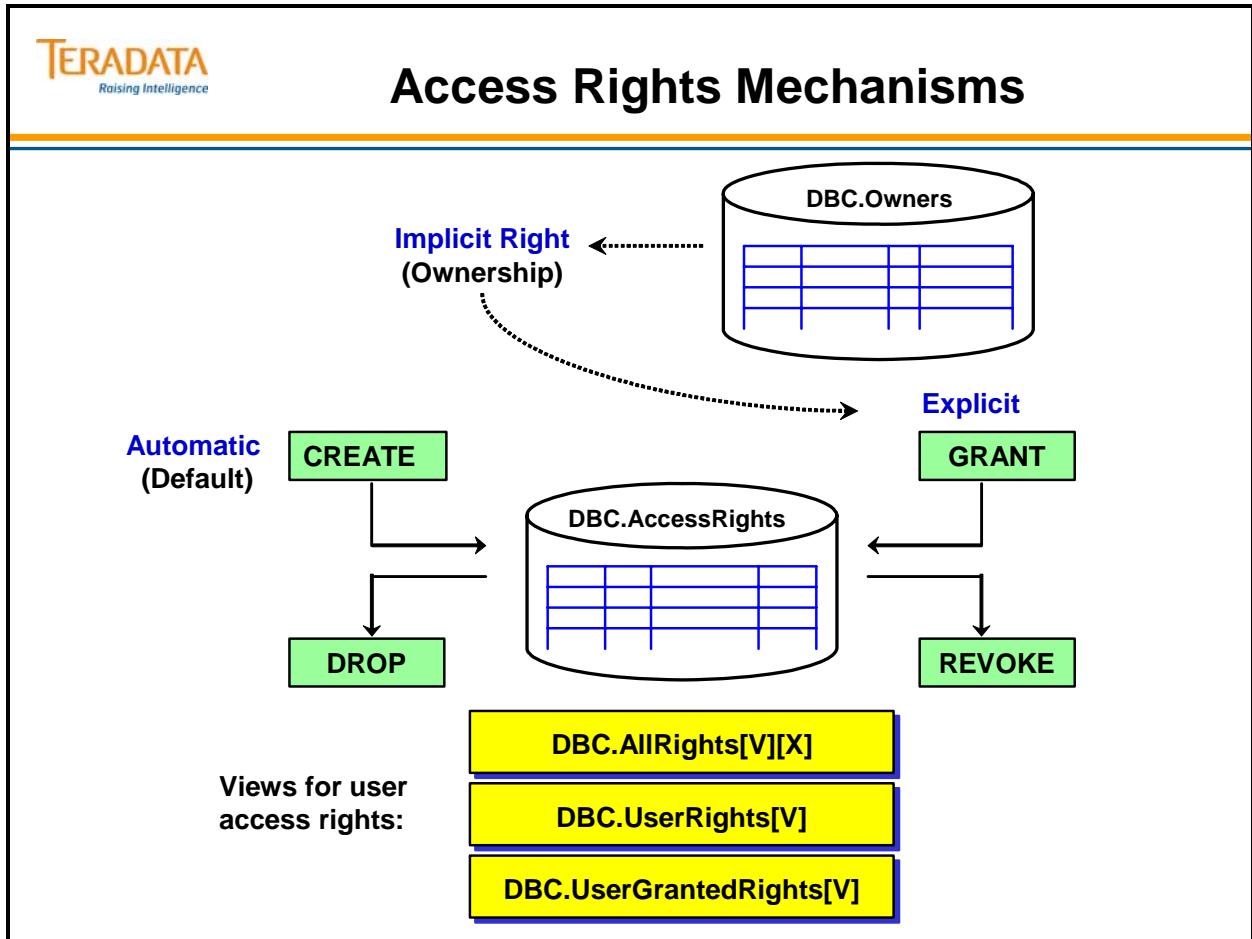
Owners (Parents) have the implicit right to grant rights on any or all of their owned objects (Children), either to themselves or to any other user or database. If an owner grants him or herself rights over any owned object, the parser will validate that GRANT statement even though the owner holds no other privileges.

Ownership rights cannot be taken away unless ownership is transferred.

Access Rights Views

The data dictionary contains three system views that return information about access rights:

- DBC.AllRights and DBC.UserRights
- DBC.UserGrantedRights
- DBC.AllRoleRights and DBC UserRoleRights



CREATE TABLE – Automatic Rights

The SQL request is preceded by the modifier EXPLAIN. As a result, the parser prints out the AMP steps (in simple English) that the CREATE statement generates.

In step 4, parallel step 10 on the facing page, you can see how the system adds each access right to the AccessRights table.

The following access rights are inserted; each with the grant authority:

- SELECT (R)
- INSERT (I)
- UPDATE (U)
- DELETE (D)
- DROP TABLE (DT)
- INDEX (IX)
- REFERENCES (RF)
- CREATE TRIGGER (CG)
- DROP TRIGGER (DG)
- DUMP (DP)
- RESTORE (RS)

If a view is created, 5 access rights are added.

Creation of a macro causes 2 access rights to be added.



CREATE TABLE – Automatic Rights

```
EXPLAIN CREATE TABLE TFACT.Customer
  (Customer_Number INTEGER, Last_Name CHAR(30),
   First_Name CHAR(20), Social_Security INTEGER)
UNIQUE PRIMARY INDEX (Customer_Number)
UNIQUE INDEX (Social_Security);
```

- 1) First, we lock TFACT.Customer 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.ArchiveLoggingObjsTbl 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.Indexes for write on a RowHash, we lock DBC.DBase for read on a RowHash, and we lock DBC.AccessRights for write on a RowHash.
- 4) We execute the following steps in parallel.
 - 1) We do a single-AMP ABORT test from DBC.ArchiveLoggingObjsTbl by way of the primary index.
 - 2) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.
 - 3) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index.
 - 4) We do an INSERT into DBC.TVFields (no lock required).

:

8) We do an INSERT into DBC.Indexes (no lock required).
 - 9) We do an INSERT into DBC.Indexes (no lock required).
 - 10) We do an INSERT into DBC.TVM (no lock required).
 - 11) We INSERT default rights to DBC.AccessRights for TFACT.Customer.
- 5) We create the table header.
- 6) We create the index subtable on TFACT.Customer.
- 7) We modify the table header TFACT.Customer.
- 8) 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.

CREATE USER – Automatic Rights

When you create a new user or database, the system automatically generates access rights for the created object and the creator of the object. The system inserts this rights information into the DBC.AccessRights table when you submit a CREATE request. You can remove these rights from the DBC.AccessRights table with the REVOKE statement.

Example

In the example on the facing page, user SYSDBA logs on to the system and creates a new user called Accounting. Both SYSDBA and Accounting have the following privileges written into the DBC.AccessRights table:

CREATE TABLE	DROP TABLE
CREATE VIEW	DROP VIEW
CREATE MACRO	DROP MACRO
CREATE TRIGGER	DROP TRIGGER
SELECT	INSERT
UPDATE	DELETE
EXECUTE	DROP PROCEDURE
CHECKPOINT	RESTORE
DUMP	DROP FUNCTION
CREATE AUTHORIZATION	DROP AUTHORIZATION

Note: CREATE and DROP AUTHORIZATION access rights are new with Teradata V2R6.1

In addition, user SYSDBA has the following rights over Accounting as its creator:

- CREATE Database/User
- DROP Database/User



CREATE USER – Automatic Rights

By issuing a CREATE USER statement, the CREATOR causes Automatic rights to be generated for both the created user and the creator.



Both SYSDBA and Accounting are given the following rights over Accounting:

CREATE Table	DROP Table	CREATE View	DROP View
CREATE Macro	DROP Macro	CREATE Trigger	DROP Trigger
SELECT	INSERT	UPDATE	DELETE
EXECUTE	DROP Procedure	DROP Function	DUMP
RESTORE	CHECKPOINT	CREATE Authorization	DROP Authorization

SYSDBA is given the following additional rights over Accounting:

CREATE Database	DROP Database	CREATE User	DROP User
-----------------	---------------	-------------	-----------

Implicit, Automatic, and Explicit Rights

Implicit rights belong to the owners of objects. Owners do not require rows in the DBC.AccessRights table to grant privileges on owned objects. Ownership rights cannot be “revoked.” An owner has the implicit right to GRANT privileges over any owned object.

When you submit a CREATE statement, the system automatically adds new rows to the DBC.AccessRights table. You can remove automatic rights with the REVOKE or DROP statements.

GRANT and REVOKE statements control explicit rights. The GRANT statement adds new rows to the DBC.AccessRights table. The REVOKE statement removes them.

Example

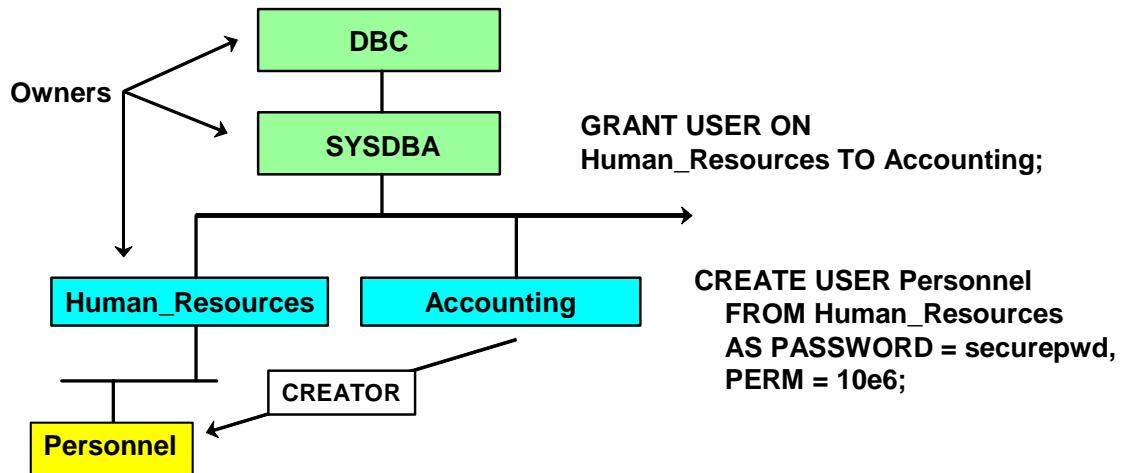
In the example, Accounting is the creator. The system automatically inserts rows for access rights in DBC.AccessRights for the creator (Accounting) and for the created user (Personnel). These rights can be revoked.

The user named Personnel is the created object. The database Personnel automatically receives all but four access rights on itself. These rights are inserted *automatically* in DBC.AccessRights. These rights can be revoked.

The user named Human_Resources is the immediate owner. The system does not insert any rows in the Data Dictionary for Human Resources. However, Human_Resources has the owner's implicit right to grant itself rights over Personnel. You cannot revoke the right to GRANT (or re-GRANT) rights over owned objects.



Implicit, Automatic, and Explicit Rights



How many automatic access rights are created for Personnel?

How many automatic access rights are created for Human_Resources?

How many automatic access rights are created for Accounting?

GRANT Command

You can use the GRANT statement to give to one or more users or databases one or more privileges on a database, user, table, view, or macro.

To grant a privilege, you must:

- Have the privilege itself and have GRANT authority
OR
- Be an owner

The recipient of an explicitly granted privilege may be:

- username The specific user(s) or database(s) named
- PUBLIC Every user in the DBC system (same as ALL DBC)
- ALL username The named user and ALL descendants

Access rights that a new user inherits because the ALL or PUBLIC option was used are referred to as “inherited rights”.

The **WITH GRANT OPTION** confers on the recipient “Grant Authority”. The recipient (or “Grantee”), holding this authority, may then grant the access right to other users, databases, or roles.

Syntax for REFERENCES or UPDATE access right for a column:

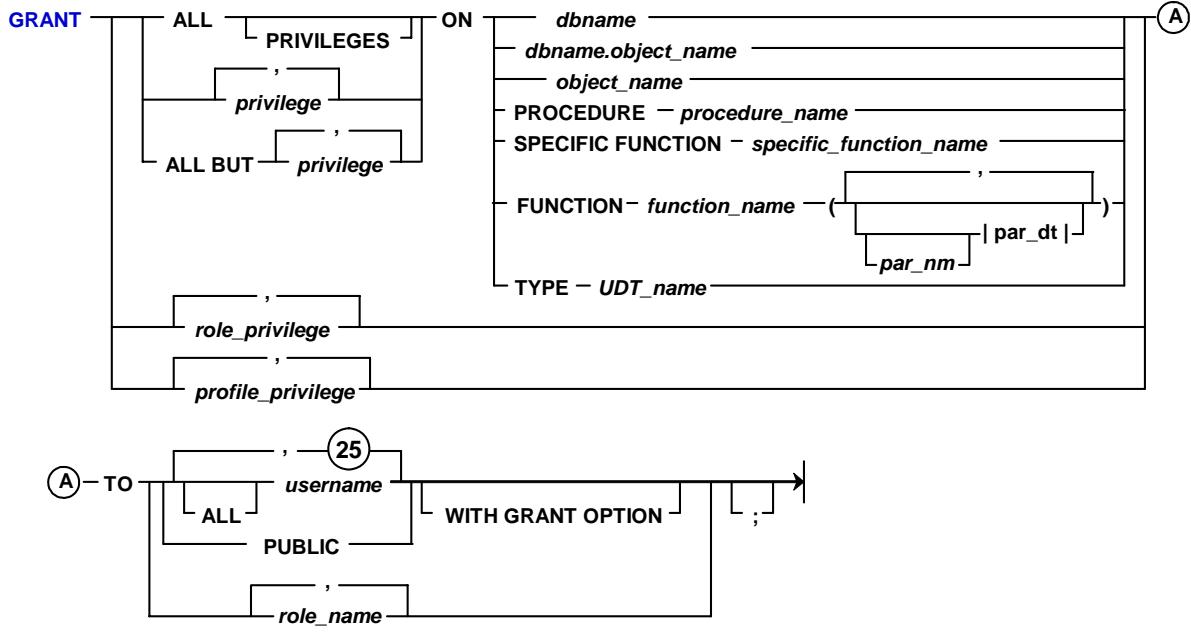
GRANT REFERENCES [(columnname list or
ALL BUT column_name_list)] ...

GRANT UPDATE [(columnname list or
ALL BUT column_name_list)] ...

GRANT Command (SQL Form)

To GRANT a privilege, the user (grantor) must have one of the following:

- Have the privilege granted, and hold GRANT authority on the privilege
- Be an owner of the object.



Granting Rights at Database Level

The facing page illustrates privileges granted at the database level.

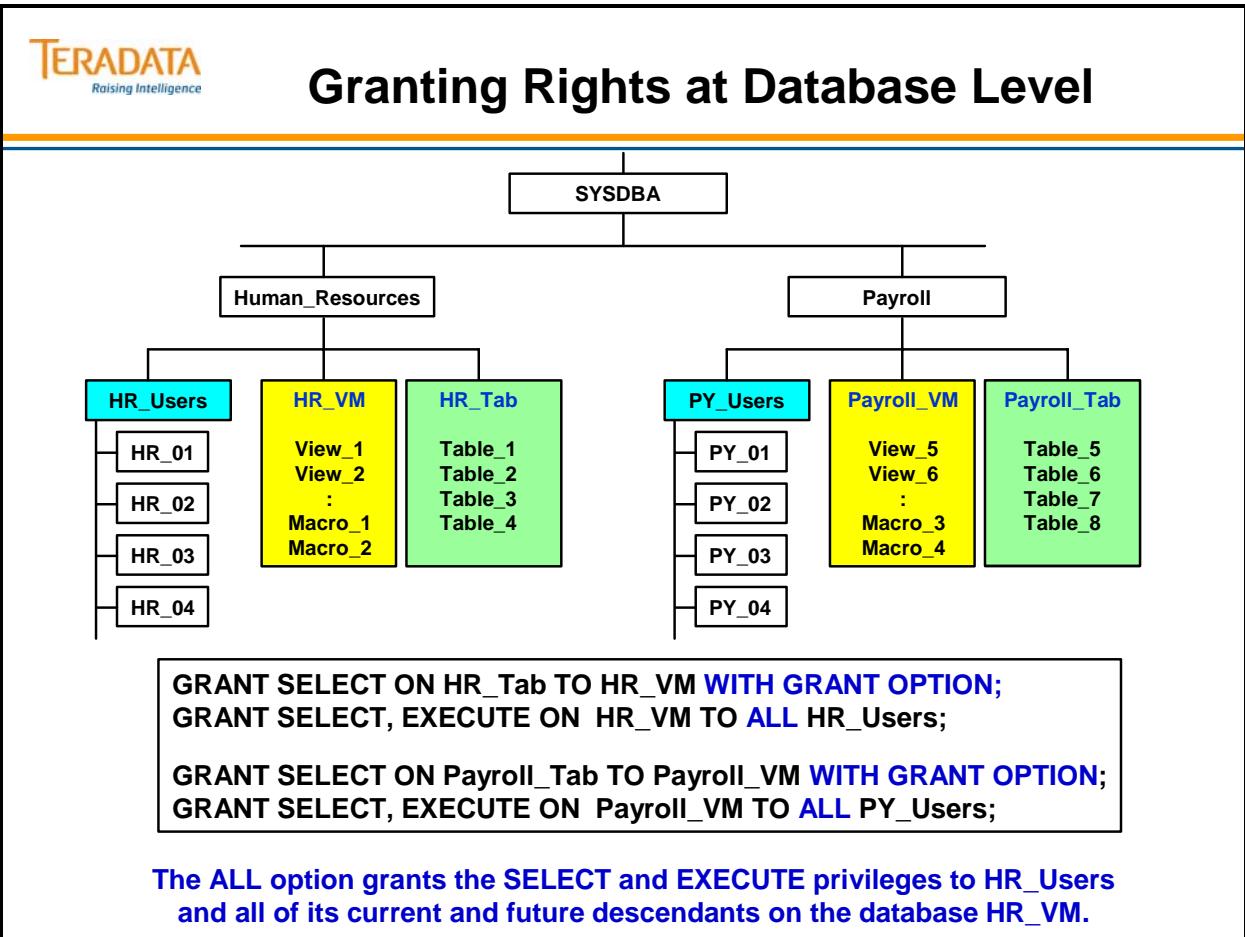
A system structure for the Teradata database is shown on the facing page and this hierarchy will be used in numerous examples.

Keys to the hierarchy on the facing page are:

- HR_Users – users that require SELECT and EXECUTE access rights on the views and macros in the HR_VM database.
- PY_Users – users that require SELECT and EXECUTE access rights on the views and macros in the Payroll_VM database.

The database HR_VM will have the SELECT WITH GRANT OPTION access right on the database named HR_Tab.

The database Payroll_VM will have the SELECT WITH GRANT OPTION access right on the database named Payroll_Tab.



GRANT Rights at the Table or Column Level

UPDATE and **REFERENCES** privileges may be granted at the table level or at the individual column(s) level.

The **INDEX** privilege must be granted at the table level, to permit the creating of secondary indexes.

Access Rights and Triggers

To create or replace a trigger, specific access rights are required.

Access Rights to Create Triggers:

- **CREATE TRIGGER** privilege on the subject table or the database.
- **SELECT** privilege on any column referenced in a **WHEN** clause or a triggered SQL statement subquery.
- **INSERT**, **UPDATE**, or **DELETE** privileges on the triggered SQL statement target table, depending on the triggered SQL statement.

Access Rights to Replace Triggers:

- **DROP TRIGGER** privilege on the subject table or the database. The exception is when you use the **REPLACE TRIGGER** statement when no target trigger exists and you instead create a new trigger.
- **SELECT** privilege on any column referenced in a **WHEN** clause or a triggered SQL statement subquery.
- **INSERT**, **UPDATE**, or **DELETE** privileges on the triggered SQL statement target table, depending on the triggered SQL statement.

Example: **CREATE TRIGGER trigger1**
 AFTER UPDATE OF (col1) ON table1 FOR EACH ROW
 WHEN NEW col1 > 100
 INSERT INTO log_table VALUES ...



GRANT Rights at the Table or Column Level

To UPDATE a table or columns of a table:

GRANT UPDATE	ON Employee TO tfact01;
GRANT UPDATE (salary_amount)	ON Employee TO tfact01;
GRANT UPDATE (ALL BUT salary_amount)	ON Employee TO tfact01;

To CREATE or ALTER a table with foreign key references:

GRANT REFERENCES	ON Employee TO tfact01;
GRANT REFERENCES (employee_number)	ON Employee TO tfact01;
GRANT REFERENCES (ALL BUT employee_number)	ON Employee TO tfact01;

The INDEX privilege is granted at the table level to allow a user to CREATE or DROP indexes on a table:

GRANT INDEX ON Employee TO tfact01;

REVOKE Command

REVOKE is passive in that it:

- Does *not* add rows to DBC.AccessRights.
- Removes rows from the DBC.AccessRights table *only* if the privileges specified exist.
- Does *not* cascade through the hierarchy unless you specify the “ALL username” option.
- Is not automatically issued for privileges granted by a grantor dropped from the system.

The REVOKE statement removes rights inserted in the DBC.AccessRights table by a CREATE statement. It can also remove explicit rights inserted in the DBC.AccessRights table by the GRANT statement.

REVOKE Recipients

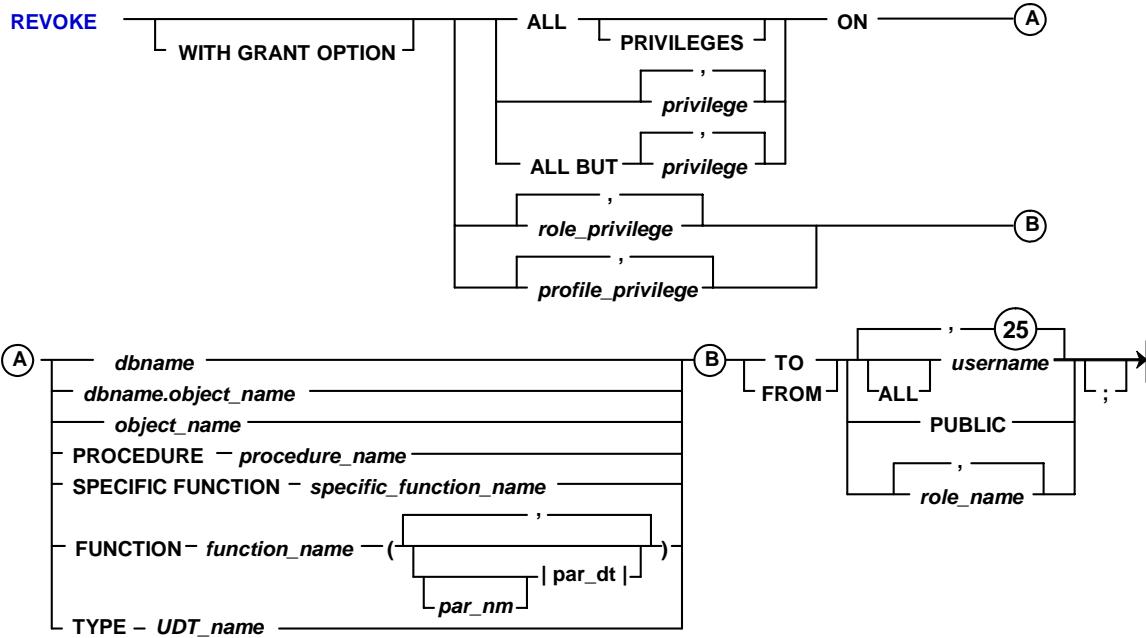
The REVOKE statement can remove privileges from one of the following:

- username A specific named user(s)
- PUBLIC Every user in the DBC system
- ALL username The named user and all of his descendants

REVOKE Command (SQL Form)

To REVOKE a privilege, the user must have one of the following:

- Have the privilege granted, and hold GRANT authority on the privilege
- Be an owner of the object.



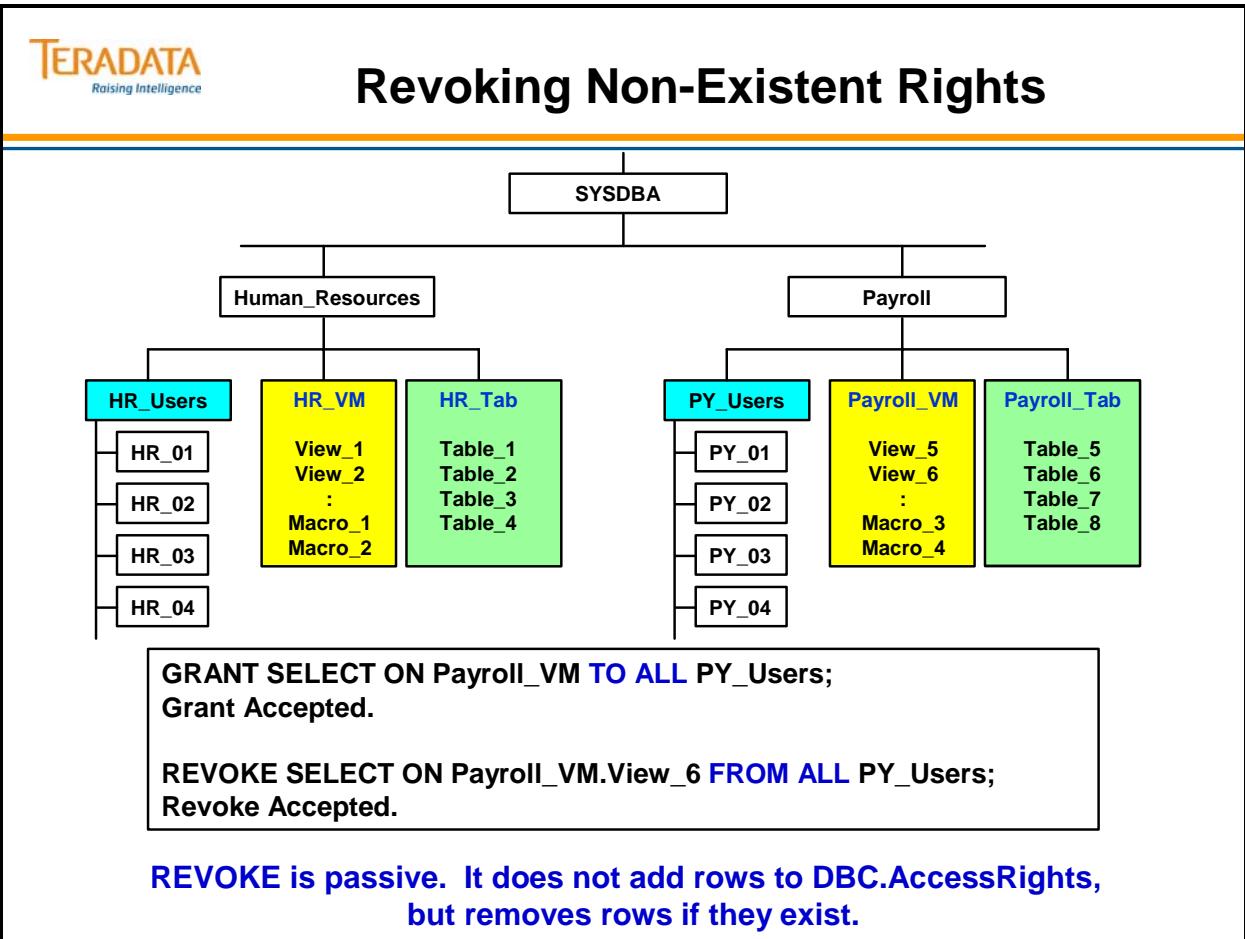
Revoking Non-Existent Rights

A REVOKE statement at the object level cannot remove privileges from that object that were granted at the database or user level because there is no correlating row in the DBC.AccessRights table for the individual object.

Example

The diagram on the facing page illustrates privileges granted at the database level. User Payroll logs on to the system, and grants the SELECT privilege to user PY_Users and ALL of its descendants on the database Payroll_VM.

Later, Payroll REVOKEs the SELECT privilege from ALL PY_Users only on View_6 that resides in Payroll_VM. Although the system returns the message “Revoke Accepted,” nothing actually happened. The user PY_Users and its descendants still have the SELECT privilege on all views residing in database Payroll_VM because the DBC.AccessRights table does not have a row correlating to View_6. Since the row granting select at the database level is still intact, all access rights remain in effect.



Removing a Level in the Hierarchy

The example on the facing page demonstrates how to remove a level from an existing hierarchy. In the first diagram, user A is the owner of users B, C, and D. User A no longer needs user B. He wants to keep users C and D.

Transfer Ownership

The first thing user A needs to do is transfer ownership of user C to A. When user A submits the GIVE statement, both user C and user D will be transferred. That is because the GIVE statement transfers the named object and all of its children. Since user D is a child of user C, both objects are transferred under user A.

Delete User or Database

In order to DROP user B, user A must first delete all objects from user B. The DELETE USER command will delete all data tables, views, triggers, stored procedures, and macros from a database or user. This command will not remove a Permanent Journal, Hash Indexes, or Join Indexes from a user or database.

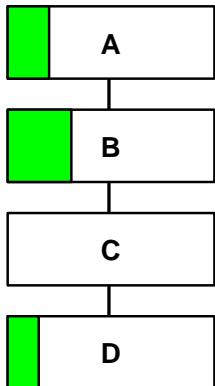
Drop User

After user A removes all objects from user B, user A can submit the DROP statement.

Access Rights

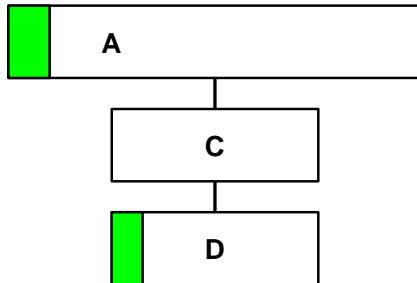
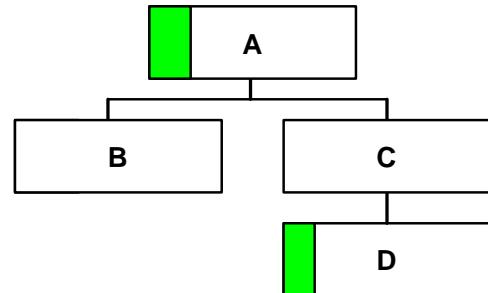
The privileges for user C and user D remain intact. Although user B, their original creator, no longer exists, the privileges granted or caused to be granted are not automatically revoked. Note that user A has recovered the perm space held by user B.

Removing a Level in the Hierarchy



LOGON with the required privileges, and

- 1) GIVE C TO A;
- 2) DELETE USER B;
- 3) DROP USER B;



Although B no longer exists as a user, the privileges granted or caused to be granted are not automatically revoked.

Inheriting Access Rights

You may inherit access rights by the placement of your user in the hierarchy. As an administrator, you can set up access rights so that any new object added to an existing user or database inherits specific access rights. Doing so saves time since you do not need to submit a GRANT statement each time you add a new user.

The immediate owner (user or database) of a view or table that is referenced by another must have the right on the referenced object that is specified (SELECT, EXECUTE, etc.) and must have that right with the GRANT option.

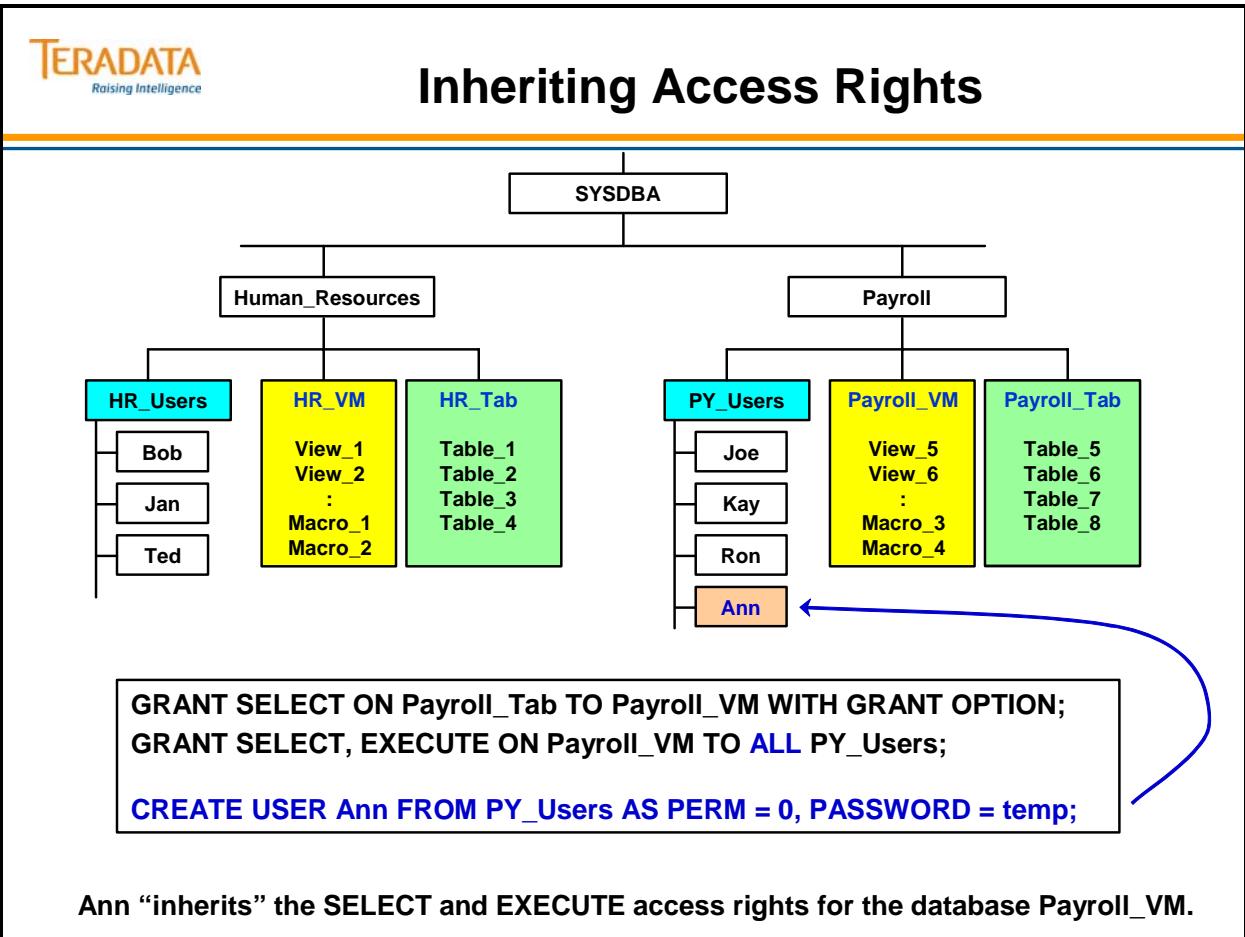
Example

The example on the facing page illustrates a user inheriting access rights.

The user Human_Resources logs on the system and grants the SELECT and EXECUTE privileges to user HR_Users and all of its current and future descendants on the database HR_VM.

The user Payroll also logs on the system and grants the SELECT and EXECUTE privileges to user PY_Users and all of its current and future descendants on the database Payroll_VM.

Later, Payroll creates a new user called Ann from the space owned by user PY_Users. Ann inherits the SELECT and EXECUTE privileges on database Payroll_VM database.



The GIVE Statement and Access Rights

When you give a user to another owner, privileges are not altered. **The GIVE statement does not alter DBC.AccessRights.** No rights on the given database or user are granted to the new ownership hierarchy as a result of the GIVE statement. The database or user that you GIVE does not receive any access rights from its new owner. The new owner gains *implicit* access rights over the transferred object and the old owner loses them.

Example

In the example on the facing page, Sysdba logs on to the system and gives user Ann to HR_Users. Ann retains the privileges that she inherited from PY_Users when she was created. Ann does not inherit any access privileges from the new owner, HR_Users, or from Human_Resources

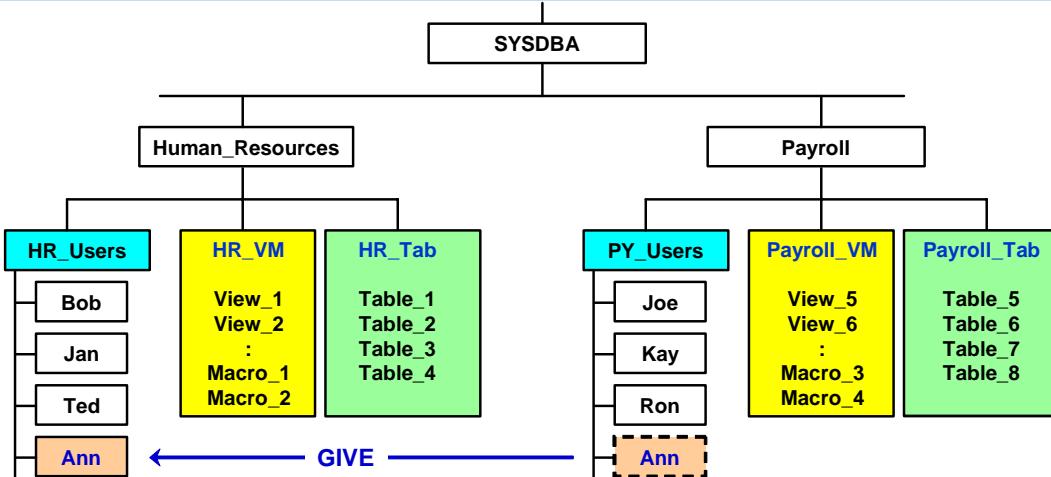
HR_Users is Ann's new owner. It has ownership rights over Ann. PY_Users loses ownership rights over Ann when she is transferred.

The syntax of the GIVE statement is as follows:

GIVE *database_name* TO *recipient_name*;



The GIVE Statement and Access Rights



NOT Recommended

```
.LOGON sysdba, password;
GIVE Ann TO HR_users;
```

The GIVE command transfers ownership, but does not change any access rights.

Recommended

```
.LOGON sysdba, password;
DROP USER Ann;
CREATE USER Ann FROM HR_Users ...;
```

The DROP will cause Ann's access rights to be removed for Payroll_VM. The CREATE will allow Ann to inherit access rights for HR_VM.

Access Rights and Views

Views that reference other views are sometimes called nested views. Views may be nested up to ten levels with V2R3 and 64 levels starting with V2R4.

View names are fully expanded (resolved) at creation time.

The system checks access rights at creation time, and validates them again at execution time. Any database referenced by the view requires access rights on all objects accessed by the view.

The facing page shows an example of a nested view.

You can create a view with the intention of read access only, or for controlled UPDATES use. For read access, the SELECT right is needed. For updates, the UPDATE right is needed.

For other users to access a view, the owner must grant the appropriate rights on the view and must have the appropriate rights WITH GRANT OPTION.

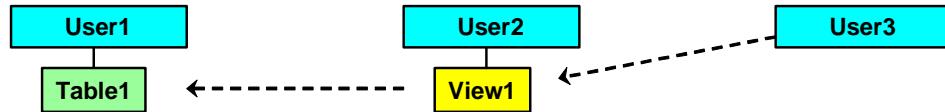
The system verifies that the creator has the appropriate right on the objects being referenced when a view is created. It also verifies that the creator has the rights needed to execute the statements defined in a macro. To grant to another user any privilege on a view or macro that references objects owned by a third user, the owner of the view or macro must have the appropriate rights with GRANT OPTION.

Teradata also verifies that the appropriate privileges exist on the objects being referenced for any user who attempts to access a view or execute a macro. This ensures that a change to a referenced object does not result in a violation of access rights when the view or macro referencing that object is invoked.



Access Rights and Views

- View names are fully expanded (resolved) at creation time.
- The system checks access rights at creation time, and validates them again at execution time.



`GRANT SELECT
ON Table1 TO User2;`

`CREATE VIEW View1
AS SELECT ...
FROM User1.Table1;
Success`

`SELECT * FROM User1.Table1;
Fails - Error 3523`

`SELECT * FROM View1;
Success`

`SELECT * FROM User2.View1;
Fails - Error 3523`

`GRANT SELECT
ON View1 TO User3;
Fails - Error 3523`

User does not have the SELECT access right on Table1 or View1.

`GRANT SELECT
ON Table1 TO User2
WITH GRANT OPTION;`

`GRANT SELECT
ON View1 TO User3;
Success`

`SELECT * FROM User2.View1;
Success`

`SELECT * FROM User1.Table1;
Fails - Error 3523`

3523 An owner referenced by the user does not have [Access right] access to [Database.Object].

Access Rights and Nested Views

Views that reference other views are sometimes called nested views. Views may be nested up to ten levels with V2R3 and 64 levels starting with V2R4. View names are fully expanded (resolved) at creation time.

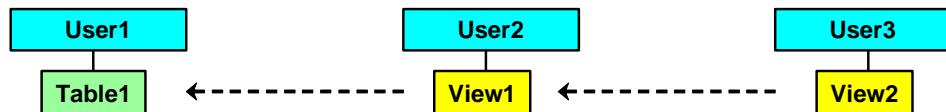
The system checks access rights at creation time, and validates them again at execution time. Any database referenced by the view requires access rights on all objects accessed by the view.

The previous example is continued on the facing page.



Access Rights and Nested Views

- Views that reference other views are sometimes called nested views. Views may be nested up to 10 levels with Release V2R3 and 64 levels starting with Release V2R4.
- The system checks access rights at creation time, and validates them again at execution time.



**GRANT SELECT
ON Table1 TO User2
WITH GRANT OPTION;**

User2 can select from Table1
and can create and use
views that access Table1.

User3 is given SELECT access
on View1 and can create View 2.
User3 can access Table1 via
View1 or View2.

**GRANT SELECT
ON View1 TO User3
WITH GRANT OPTION;
Success**

User3 can GRANT SELECT
access to View2 to other users.

**REVOKE GRANT OPTION
FOR SELECT
ON Table1 FROM User2;**

**SELECT * FROM View1;
Success**

**SELECT * FROM View2;
Fails - Error 3523**

If you REVOKE access rights from any user in the chain, the
system issues the following message:

**3523 An owner referenced by the user does not have
[Access right] access to [Database.Object].**

System Views for Access Rights

There are three system views you can use to obtain information about access rights. (These views access the DBC.AccessRights table to obtain needed information.) They are:

- DBC.AllRights[V][X]
- DBC.UserRights[V]
- DBC.UserGrantedRights[V]

DBC.AllRights[V][X]

The DBC.AllRights[X] views provide information about all rights that have been automatically or explicitly granted.

DBC.UserRights[V]

This view provides information about all rights that the user has acquired, either automatically or explicitly.

DBC.UserGrantedRights[V]

This view provides information about rights that the current user has explicitly granted to other users.



System Views for Access Rights

View

Description

DBC.AllRights[V][X]

Provides information about all rights that have been automatically or explicitly granted.

DBC.UserRights[V]

Provides information about all rights the user has acquired, either automatically or explicitly.

DBC.UserGrantedRights[V]

Provides information about rights which the current user explicitly has granted to other users.

DBC.AllRights[X] and DBC.UserRights Views

Examples of access rights and their abbreviations include:

DATABASE	=	CREATE (CD)	+	DROP (DD)
USER	=	CREATE (CU)	+	DROP (DU)
TABLE	=	CREATE (CT)	+	DROP (DT)
VIEW	=	CREATE (CV)	+	DROP (DV)
MACRO	=	CREATE (CM)	+	DROP (DM)
TRIGGER	=	CREATE (CG)	+	DROP (DG)
PROCEDURE	=	CREATE (PC)	+	DROP (PD)
FUNCTION	=	CREATE (CF)	+	DROP (DF)
INDEX		Table		IX
REFERENCES		Table or Column		RF
SELECT/RETRIEVE		Table or View		R
UPDATE		Table or View or Table Column or View Column		U
INSERT		Table or View		I
DELETE		Table or View		D
EXECUTE		Macro		E
CHECKPOINT		Journal Tables		CP
DUMP		Data and/or Journal Tables		DP
RESTORE		Data and/or Journal Tables		RS
CREATE ROLE		Create a role		CR
DROP ROLE		Drop a role		DR
CREATE PROFILE		Create a profile of user attributes		CO
DROP PROFILE		Drop a profile of user attributes		DO
CREATE PROCEDURE		Stored procedure		PC
DROP PROCEDURE		Stored procedure		PD
EXECUTE PROCEDURE		Stored procedure		PE
SET SESSION RATE				SS
SET RESOURCE RATE				SR
ABORT SESSION				AS
MONITOR RESOURCE				MR
MONITOR SESSION				MS

The RESTORE statement also allows the recipient to execute ROLLBACK, ROLLFORWARD, and DELETE JOURNAL commands in the ARC facility. The DROP allows COMMENT ON and COLLECT STATISTICS on the object.



AllRights and UserRights Views

Provides information about the objects on which all users (DBC.AllRights), or the current user (DBC.UserRights), have automatically or explicitly been granted privileges.

DBC.AllRights[V][X]

UserName	DatabaseName
TableName	ColumnName
AccessRight	GrantAuthority
GrantorName	AllnessFlag
CreatorName	CreateTimeStamp

DBC.UserRights[V]

DatabaseName	TableName
ColumnName	AccessRight
GrantAuthority	GrantorName
CreatorName	CreateTimeStamp

Example:

All rights held by the user at the database level (for user tfact07).

```
SELECT DatabaseName (FORMAT 'X(16)')
,AccessRight
,GrantorName (FORMAT 'X(16)')
FROM DBC.UserRights
WHERE Tablename = 'ALL'
ORDER BY 1, 2;
```

Example Results:

DatabaseName	AccessRight	GrantorName
AP	R	DBC
PD	D	SYSDBA
PD	I	SYSDBA
PD	R	SYSDBA
PD	U	SYSDBA
tfact07	CG	SYSDBA

DBC.UserGrantedRights View

The DBC.UserGrantedRights view provides information about objects on which the current user has explicitly granted privileges. When you submit the GRANT statement, the system stores explicit privileges as rows in the DBC.AccessRights table.

Column definitions in this view include:

<u>Column</u>	<u>Definition</u>
---------------	-------------------

Grantee	The recipient of the access right.
---------	------------------------------------

AllnessFlag	Y (Yes) indicates the privilege was granted to all. N (No) indicates the privilege was not granted to all.
-------------	---



DBC.UserGrantedRights View

Provides information about objects on which the current user has explicitly granted privileges to other users.

DBC.UserGrantedRights[V]

DatabaseName	TableName	ColumnName	Grantee
AccessRight	GrantAuthority	AllnessFlag	CreatorName
CreateTimeStamp			

Example:

List the rights explicitly granted by the current user.

Example Results:

```
SELECT DatabaseName (FORMAT 'X(12)')
,TableName (FORMAT 'X(15)')
,Grantee (FORMAT 'X(10)')
,AccessRight
,AllnessFlag
FROM DBC.UserGrantedRights
ORDER BY 1, 2, 3, 4;
```

DatabaseName	TableName	Grantee	AccessRight	AllnessFlag
AP	All	tfact07	R	N
DS	Daily_Sales	tfact03	R	N
DS	Daily_Sales	tfact03	RF	N
DS	Order_Item_JI	tfact03	IX	N
PD	All	Students	R	Y

Teradata Administrator – Grant/Revoke Rights

The facing page contains an example of the Grant/Revoke dialog box that is provided when using the menus of Teradata Administrator.

Tools ➔ Grant/Revoke ➔ Object Rights

The help facility of Teradata Administrator also lists all of the Access Right Codes.

TERADATA
Raising Intelligence

Teradata Administrator GRANT/REVOKE Rights

Teradata Administrator can be used to easily grant or revoke access rights.

Tools → Grant/Revoke → Object Rights

Grant / Revoke Objects [tdt5-1]

Database Name	To / From User	Role	Public
AP	TDPUSER	tdwm	<input type="checkbox"/>
	Teradata_Classes		<input type="checkbox"/>
	Teradata_Training		<input type="checkbox"/>
	TFAC		<input type="checkbox"/>
	tfact01		<input checked="" type="checkbox"/>
	tfact02		<input checked="" type="checkbox"/>
	tfact03		<input checked="" type="checkbox"/>
	THF		<input type="checkbox"/>
	tjk01		<input type="checkbox"/>
	tjk02		<input type="checkbox"/>
	tjk03		<input type="checkbox"/>
	<input type="checkbox"/> And All Children		

Normal

<input checked="" type="checkbox"/> Execute	<input type="checkbox"/> Index
<input checked="" type="checkbox"/> Select	<input type="checkbox"/> References
<input type="checkbox"/> Insert	<input type="checkbox"/> Dump
<input type="checkbox"/> Update	<input type="checkbox"/> Restore
<input type="checkbox"/> Delete	<input type="checkbox"/> Checkpoint
<input type="checkbox"/> Exec Procedure	<input type="checkbox"/> Alter Procedure
<input type="checkbox"/> Execute Function	<input type="checkbox"/> Alter Function
<input type="checkbox"/> UDT Usage	<input type="checkbox"/> Alter External Proc

Create

<input type="checkbox"/> Table	<input type="checkbox"/> View
<input type="checkbox"/> Macro	<input type="checkbox"/> Macro
<input type="checkbox"/> Database	<input type="checkbox"/> Database
<input type="checkbox"/> User	<input type="checkbox"/> User
<input type="checkbox"/> Trigger	<input type="checkbox"/> Trigger
<input type="checkbox"/> Procedure	<input type="checkbox"/> Procedure
<input type="checkbox"/> Function	<input type="checkbox"/> Function
<input type="checkbox"/> External Proc	<input type="checkbox"/> External Proc
<input type="checkbox"/> Authorization	<input type="checkbox"/> Authorization

Drop

<input type="checkbox"/> All	<input type="checkbox"/> Dictionary
<input type="checkbox"/> Table	<input type="checkbox"/> Create
<input type="checkbox"/> View	<input type="checkbox"/> Drop
<input type="checkbox"/> Macro	<input type="checkbox"/> Access
<input type="checkbox"/> Database	<input type="checkbox"/> All But
<input type="checkbox"/> User	<input checked="" type="checkbox"/> Grant!
<input type="checkbox"/> Trigger	
<input type="checkbox"/> Procedure	
<input type="checkbox"/> Function	
<input type="checkbox"/> External Proc	
<input type="checkbox"/> Authorization	

Buttons: Grant, Revoke, Display, Clear, Close

Teradata Administrator – Rights on DB/User

The facing page contains an example of using Teradata Administrator to view the access rights that are on a specific database or user.

TERADATA
Raising Intelligence

Teradata Administrator Rights on DB/User

Teradata Administrator can also be used to easily view existing access rights

The screenshot shows the Teradata Administrator interface with the title bar "Teradata Administrator - [tdt5-1.DBC]". The main window displays a tree view of database objects under "Teradata_Training", including "Guest_Users", "Instructors", "Students", and "Teradata_Class". A context menu is open over the "AP" node in the tree, with "Rights on DB/User" selected. This menu also includes options like "List All Objects", "List Tables & Indexes", "List Views", "List Macros & Procedures", and "Database Information". To the right of the tree view is a table titled "Rights on DB/User - AP" with columns: AccessRight, GrantAuthority, GrantorName, and AllnessFlag. The table lists various rights assigned to the "TERADATA_TRAINING" grantor. The status bar at the bottom right shows "Elapsed 00:00:01".

**Right-click on the database AP and select the option.
In this example, Rights on DB/User was selected.**

Access Rights Summary

The facing page summarizes some important concepts regarding this module.



Access Rights Summary

- Access Rights (Privileges) are maintained in the data dictionary.
- Rows are inserted into or removed from DBC.AccessRights by:
 - CREATE or DROP statements
 - GRANT or REVOKE statements
- Creators are given automatic rights on created objects.
- A newly created user or database is given all rights on themselves except:
 - CREATE Database/User
 - DROP Database/User
- Owners have the right to grant privileges on their owned objects.
- The GIVE command affects ownership, but not information in the DBC.AccessRights table.

Review Questions

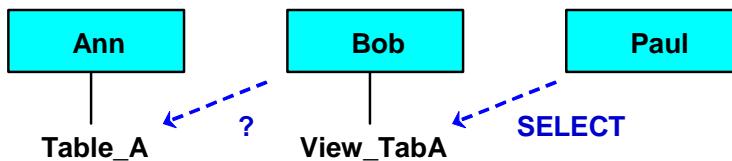
Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. True or False There are only two types of access rights or privileges: explicit and implicit.
2. True or False The primary statements you use to manage access rights are GRANT, REVOKE, and GIVE.
3. The _____ option on the GRANT command grants privileges to a database or user and all of its current and future descendants.
4. The _____ and _____ access rights can be granted at the column level.
5. The _____ user is used to grant an access right to every user in the system.
6. Given the following: Ann owns Table_A, Bob creates View_TabA and grants SELECT on View_TabA to Paul.

What access right does Ann give Bob on Table_A so Paul can use View_TabA to access Table_A?



Teradata Training

Notes

Module 46



Roles and Profiles

After completing this module, you should be able to:

- List 3 advantages of utilizing roles and profiles.
- Identify the access rights needed to create roles and profiles.
- Number the steps of access right validation.
- Specify the order of precedence of user/session parameters.
- Use roles and profiles when creating new users.
- Use system views to display role and profile information.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

What are Roles and Profiles?	46-4
Advantages of Roles	46-6
Access Rights without Roles.....	46-8
Access Rights Issues (prior to Roles).....	46-8
Access Rights Using a Role	46-10
Implementing Roles	46-12
Current or Active Roles	46-14
Nesting of Roles.....	46-16
Example of Using “Nested Roles”.....	46-18
Access Rights Validation and Roles	46-20
SQL Statements to Support Roles.....	46-22
GRANT Command (SQL Form)	46-24
REVOKE Command (SQL Form).....	46-26
GRANT and REVOKE Commands (Role Form).....	46-28
System Hierarchy (used in following examples)	46-30
Example of Using Roles.....	46-32
Example of Using Roles (cont.)	46-34
Example of Using Roles (cont.)	46-36
RoleInfo View.....	46-38
RoleMembers View	46-40
DBC.AccessRights and “Rights” Views.....	46-42
AllRoleRights and UserRoleRights Views	46-44
Steps to Implementing Roles	46-46
Profiles	46-48
Example of Simplifying User Management.....	46-50
Implementing Profiles	46-52
Impact of Profiles on Users.....	46-54
CREATE/MODIFY PROFILE Statement	46-56
Password Attributes (CREATE/MODIFY PROFILE)	46-58
Teradata Password Control	46-60
Teradata Password Control (cont.).....	46-62
Teradata Password Control Options	46-64
CREATE PROFILE Example.....	46-66
Teradata Administrator CREATE PROFILE Example	46-68
CREATE PROFILE Example (cont.)	46-70
DROP PROFILE Statement.....	46-72
ProfileInfo View	46-74
Miscellaneous SQL Functions	46-76
Summary	46-78
Review Questions	46-80
Lab Exercise 46-1	46-82
Lab Exercise 46-2	46-86

What are Roles and Profiles?

Starting with Teradata V2R5.0, two new database administration and security concepts have been introduced – **roles** and **profiles**.

A role can be viewed as a pseudo-user with privileges on a number of database objects. Any user granted a role can take on the identity of the pseudo-user and access all of the objects it has rights to.

A database administrator can create different roles for different job functions and responsibilities, grant specific privileges on database objects to these roles, and then grant these roles to users.

The other new administration feature discussed in this module is profiles. With profiles, you define a set of system parameters. Assigning a profile to a group of users makes sure that all group members operate with a common set of parameters. Profiles will be discussed in the second part of this module.



What are Roles and Profiles?

Starting with Teradata V2R5, two new administration/security features were introduced – roles and profiles.

- Roles and profiles simplify the management of users and access rights.

What is a “role”?

- A role is simply a collection of access rights.
 - Rights are first granted to a role and the role is then granted to users.
- A DBA can create different roles for different job functions and responsibilities.
- Roles can help reduce the number of rows in the DBC.AccessRights table.

What is a “profile”?

- A profile is a set of common user parameters that can be applied to a group of users.
- A profile setting (e.g., SPOOL) can be changed with one command and this new value is immediately applied to every assigned user.

Advantages of Roles

Advantages of roles include:

- Simplify access rights administration

A database administrator can grant rights on database objects to a role and have these rights automatically applied to all users assigned to that role. When a user's function within his organization changes, it is easier to change his/her role than deleting old rights and granting new rights that go along with the new function.

- Reduce disk space usage

Maintaining rights on a role level rather than on an individual level makes the size of the DBC.AccessRights table much smaller. Instead of inserting one row per user per right on a database object, one row per role per right is placed in the DBC.AccessRights table.

- Better performance

Roles can improve performance and reduces dictionary contention for DDL

If roles are fully utilized on a system, roles will reduce the size of the AccessRights table and improves the performance of DDL commands that do full-file scans of this table.

- Faster DROP/DELETE USER/DATABASE, DROP TABLE/VIEW/MACRO due to shorter scans of the DBC.AccessRights table.
- Faster CREATE USER, DATABASE - remove copy of hierarchical inherited rights.
- Less dictionary contention during DDL operations because the commands use less time.



Advantages of Roles

What are the advantages of “roles”?

- Simplify access rights management by allowing grants and revokes of multiple rights with one request.
 - useful when an employee changes job function (role) within the company.
 - if a job function needs a new access right, grant it to the role and it is effective immediately.
- The number of access rights in the DBC.AccessRights table is reduced.
 - Disk space usage is reduced when rights are managed on role level rather than individual level.
- Improves performance and reduces dictionary contention for DDL, especially CREATE USER.
 - Removal of hierarchical inherited rights improves DDL performance and reduces dictionary contention.

Access Rights without Roles

The facing page illustrates the following:

- If 10 users have the SELECT access right on each of 10 views, there would be 100 rows in the DBC.AccessRights table for these 10 users.
- What if there were 50,000 users in the system and there were 500 views needed by each user? The DBC.AccessRights table would have 25 million rows.

When a new user is added in this simple example, 10 rows have to be added to the DBC.AccessRights table.

Access Rights Issues (prior to Roles)

The role concept provides a solution to the following problem.

Prior to Teradata V2R5 and the concept of roles, there are typically 2 ways of granting rights to a large user base:

1. Use the ALL option of the GRANT statement to grant rights on the shared object(s) to a parent database. Sometimes this is referred to as a “profile” database or a “group” database in V2R4.1. Do not confuse the logical profile database with the Profile capability in V2R5.

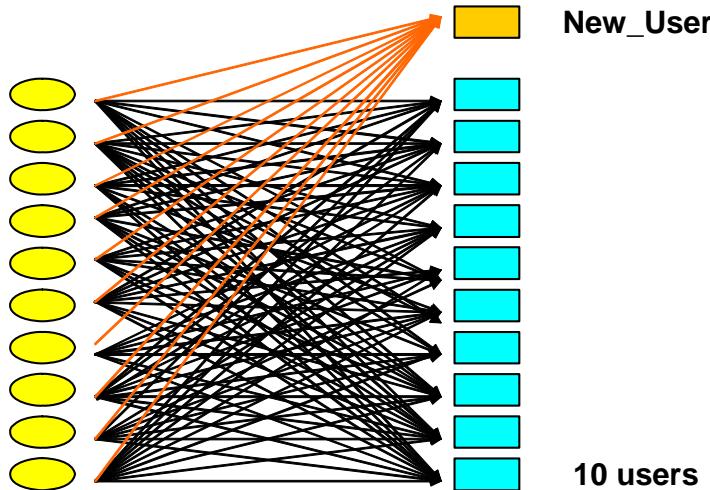
`GRANT SELECT ON database_object TO ALL profile_database;`

Then, create users under the profile database. The system will automatically grant all rights held by the profile database to each user created under the profile database. This is frequently referred to as “inherited rights”.

2. Grant the rights to users individually – an administrative nightmare.

Access Rights Without Roles

10 views
(possibly in
different
databases)



GRANT SELECT ON View1 TO New_User; GRANT SELECT ON View2 TO New_User; ...

When a new user is given the SELECT access right to these 10 views, 10 new access right rows are added to the DBC.AccessRights table.

In this simple example, these 10 views and 11 users would place 110 access right rows in the DBC.AccessRights table.

Access Rights Using a Role

When creating a new user, only one right to use a role needs to be granted, as opposed to a right for every table/view/macro/stored procedure that the user needs to access.

As mentioned earlier, a role can be viewed as a pseudo-user with privileges on a number of database objects. Any user granted a role can take on the identity of the pseudo-user and access all of the objects it has rights to.

A database administrator can create different roles for different job functions and responsibilities, grant specific privileges on database objects to these roles, and then grant these roles to users.

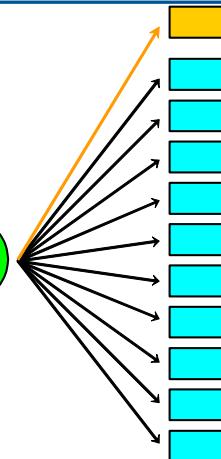
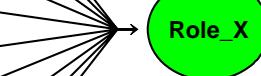
In the example on the facing page, the GRANT Role_X to New_User places a row in the DBC.RoleGrants table, not the DBC.AccessRights table.

Note:

When an access right is granted to a role, a row is placed in the DBC.AccessRights table. The DBC.AllRights system view only shows access rights associated with users, not roles. The DBC.UserRoleRights system view shows access right rows associated with roles.

Access Rights Using a Role

10 views
(possibly in
different
databases)



New_User

10 users

First, create a role and grant privileges to the role.

```
CREATE ROLE Role_X;  
GRANT SELECT ON View1 TO Role_X; GRANT SELECT ON View2 TO Role_X; ...
```

When creating a new user, only one right to use a role needs to be granted.

```
GRANT Role_X TO New_User;
```

This command places a row in the DBC.RoleGrants table, not DBC.AccessRights.

With 10 views, 1 role, and 11 users, there would be 10 rows in the DBC.AccessRights table and 11 rows in the DBC.RoleGrants table.

Implementing Roles

Roles define access privileges on database objects. When you assign a default role to a user, you give the user access to all the objects that the role has been granted privileges to. A default role that has a role as a member gives the user additional access to all the objects that the nested role has privileges to.

A newly created role does not have any associated privileges until grants are made to it. To manage user access privileges, you can:

- Create different roles for different job functions and responsibilities.
- Grant specific privileges on database objects to the roles.
- Assign default roles to users.
- Add members to the role.
- Members of a role can be users or other roles.
- Roles can only be nested one level. Thus, a role that has a role member cannot also be a member of another role.

The CREATE ROLE and DROP ROLE access rights are system rights. These rights are not on a specific database object. Note that the ROLE privileges can only be granted to a user and not to a role or database.

The example on the facing page explicitly identifies the CREATE ROLE and DROP ROLE rights for Sysdba. Another technique of granting both the CREATE ROLE and DROP ROLE access rights to Sysdba is to use the following SQL.

GRANT ROLE TO SYSDBA WITH GRANT OPTION;

The key word ROLE will give both the CREATE ROLE and DROP ROLE access rights.

Note:

If Sysdba is only given the CREATE ROLE access right, Sysdba can create new roles and Sysdba can drop roles that he/she has created. Sysdba would not be able to drop roles created by other users (such as DBC).

The syntax to create a new role is simply:

CREATE ROLE *role_name*;

When a role is first created, it does not have any associated rights until grants are made to it.



Implementing Roles

What access rights are used to create new roles?

- CREATE ROLE – needed to create new roles
- DROP ROLE – needed to drop roles

Who is allowed to create and modify roles?

- Initially, only DBC has the CREATE ROLE and DROP ROLE access rights.
- As DBC, give the “role” access rights to the database administrators (e.g., Sysdba).

GRANT CREATE ROLE, DROP ROLE TO Sysdba WITH GRANT OPTION;

How are access rights associated with a role?

- First, create a role.

CREATE ROLE HR_Role;

A newly created role does not have any associated rights until grants are made to it.

- Use the GRANT (or REVOKE) command to assign (or take away) access rights to (or from) the role.

GRANT SELECT, EXECUTE ON HR_VM TO HR_Role;

Current or Active Roles

With Teradata V2R5.0, at any time, only one role may be the session's current role. Enabled roles are the session's current role plus any nested roles. At logon time, the current role will be the user's default role.

Starting with Teradata V2R5.1, the SET ROLE ALL option is available and this allows a user to have all valid roles (for that user) to be active or available.

Create User or Modify User

The user executing the CREATE USER command with the DEFAULT ROLE option must have ADMIN privileges on a specified role. The default role is automatically granted to the newly created user.

The user executing the MODIFY USER command with the DEFAULT ROLE option must also have ADMIN privileges on a specified role. The new default role must have first be directly granted to the user before modifying the DEFAULT ROLE with the MODIFY USER command.



Current or Active Roles

How are users associated with a role?

- The role needs to be granted to the user.

`GRANT HR_Role TO user1;`

With V2R5.0, only one role can be the session's current or active role.

- Enabled roles are referred to as the **current role** plus any nested roles.
- At logon, the current role is determined by the DEFAULT ROLE value for the user.

`CREATE/MODIFY USER user1 AS ... , DEFAULT ROLE = HR_Role;`

- A user may change roles by executing the following command:

`SET ROLE role_name ;`

Starting with V2R5.1, the SET ROLE ALL command allows a user to have all valid roles (for that user) to be active.

`CREATE/MODIFY USER user1 AS ... , DEFAULT ROLE = ALL;`

or `SET ROLE ALL;` (may be specified at the session level by the user)

ANSI Note: A session that has only one current role complies with the ANSI standard.

Nesting of Roles

Roles define access privileges on database objects. When you assign a default role to a user, you give the user access to all the objects that the role has been granted privileges to. It is possible to grant a role to another role. This is referred to as “nesting”. Teradata supports one level of nesting.

If a role has another role as a member (a role has been granted to a role) and the role is the active role for a user, then a user gets additional access to all the objects that the nested role has privileges to.

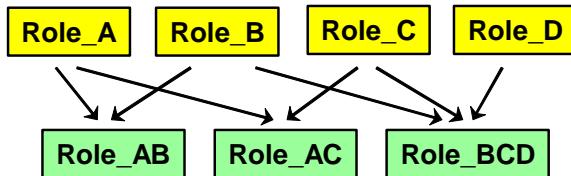
For example:

- Assume Role_A and Role_B is granted to Role_AB and assume that Role_AB is the current role of a user. The user then has the following access rights:
 - Access rights directly assigned to the user
 - Access rights assigned to Role_A
 - Access rights assigned to Role_B
 - Access rights assigned to Role_AB

Nesting of Roles

You can GRANT a role to another role – referred to as “nesting of roles”.

Allowed: 1 level of nesting

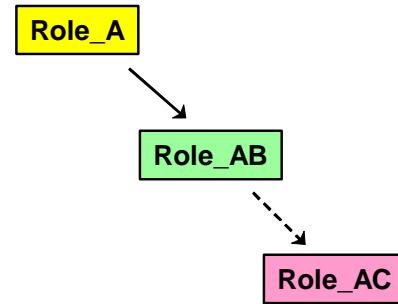


```

GRANT Role_A TO Role_AB;
GRANT Role_B TO Role_AB;
GRANT Role_A TO Role_AC;
GRANT Role_C TO Role_AC;
GRANT Role_B TO Role_BCD;
GRANT Role_C TO Role_BCD;
GRANT Role_D TO Role_BCD;
  
```

A user that is granted access to Role_AB also has all of the access rights associated with Role_A, Role_B, and Role_AC.

Not Allowed: 2nd level of nesting



```

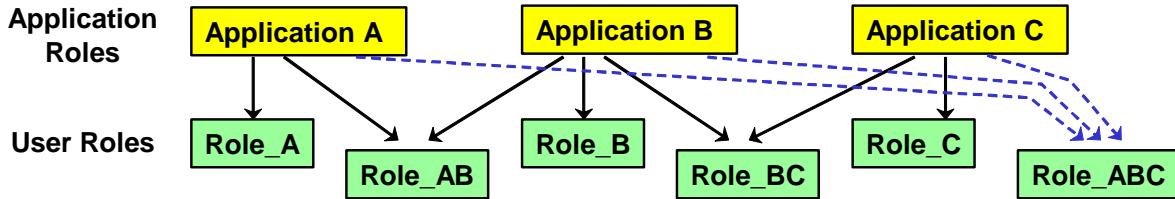
GRANT Role_A TO Role_AB; /*accepted*/
GRANT Role_AB TO Role_AC; /*fails*/
  
```

Example of Using “Nested Roles”

The facing page contains an example of using nested roles.

Example of Using "Nested Roles"

Access rights required by an Application are assigned to "application" roles.



Users are assigned to a role at this level based on job requirements.

Characteristics include:

- Users are only assigned to one "role" – ANSI standard.
- Provides a logical separation between application access rights and user access rights.
 - Access rights for an application are only assigned to a single "application" role.
 - For example, if a user needs to use Applications A, B, and C, then the user is granted access to Role_ABC.

Access Rights Validation and Roles

The validation of access rights for accessing a given database object is carried out in one or more steps. The first step verifies if a right has been granted on an individual level. If no such right exists and there is a current role for the session, then the second and third steps verify if a right has been granted to a role. The actual search goes like this:

- 1) Search the AccessRights table for a UserId-ObjectId pair entry for the required right. In this step, the system will check for rights at the database/user level and at the object (e.g., table, view) level.
- 2) If the access right is not yet found and the user has a current role, search the AccessRights table for RoleId-ObjectId pair entry for the required right.
- 3) If not yet found, retrieve all roles nested within the current role from the RoleGrants table. For each nested role, search the AccessRights table for RoleId-ObjectId pair entry for the required right.
- 4) If not yet found, check if the right is a Public right.

Performance note: If numerous roles are nested within the current role, there may have noticeable performance impact on “short requests”. A few more access right checks won't be noticed on a 1-hour query.

Notes: The following indexes are placed on the DBC.AccessRights, DBC.RoleGrants, and DBC.Roles tables and are used by Teradata software.

DBC.AccessRights

PI – (NUP1) – (UserId, DatabaseId)
SI – (NUSI) – (TVMId)

DBC.RoleGrants

PI – (NUP1) – (GranteeId)
SI – (USI) – (GranteeId, RoleId)
SI – (NUSI) – (RoleId)

DBC.Roles

PI – (UPI) – (RoleNameI)
SI – (USI) – (RoleId)



Access Rights Validation and Roles

Validation of access rights for accessing a given database object will be carried out in the following steps.

Order of access right validation is:

- 1) Check the DBC.AccessRights table for the required right **at the individual level**.
- 2) If the user has a current role, check the DBC.AccessRights table for the required right **at the role level**.
- 3) **Retrieve all roles nested within the current role from the DBC.RoleGrants table.** For each nested role, check the DBC.AccessRights table for the required right.
- 4) Check if required right is a **PUBLIC** right.

SQL Statements to Support Roles

Some miscellaneous rules concerning roles include:

- Roles may only be granted to users and other roles.
- There is no limit on the number of roles that can be granted to a grantee.
- The default role for a user will automatically be made the current role for the session when he first logs on. The default role can be established with the CREATE USER or MODIFY USER commands.
- A role grantor can only be a user, but a role grantee can be a user or another role. A role may share the same name as a profile, table, column, view, macro, trigger, or stored procedure. However, a role name must be unique amongst users, databases and roles.
- The role creator is automatically granted membership to the newly created role WITH ADMIN OPTION, which makes the role creator a member of the role who can grant membership to the role to other users and roles.

Dropping a Role

The following users can drop a role:

1. DBC
2. Any user given the system right DROP ROLE
3. Any user granted the role WITH ADMIN OPTION
4. A user whose current role has the specified role as a nested role, and the nested role was granted to the current role WITH ADMIN OPTION

The creator does not have the implicit right to drop a role. If WITH ADMIN OPTION and DROP ROLE rights are revoked from him/her, he/she will not be able to drop the role.

Default role settings for all users with the dropped role as their default role do *not* reset to NULL. Affected users receive no warnings or errors the next time they log on. The system does not use the obsolete default role for privileges validation.

If a dropped default role is subsequently recreated, it reassumes its status as the default role, but it has a different role ID number than it had before being dropped.



SQL Statements to Support Roles

Command Syntax:

CREATE ROLE *role_name*;

GRANT *access_rights* TO *role_name*;

GRANT *role_name* TO *user_name* [WITH ADMIN OPTION];

- ADMIN OPTION allows grantee the right to grant or drop the role.

SET ROLE *role_name* / NONE / NULL / ALL;

- Assigns/changes current role for session.
- Role must be granted to user before statement is valid.
- **SET ROLE ALL; (V2R5.1 option)** - All valid roles for user are available to user.

CREATE/MODIFY USER *user1* AS ..., DEFAULT ROLE = *role_name*;

- When the user logs on, the default role will become the session's initial current role.

Other commands:

REVOKE ... *role_name* ... ;

DROP ROLE *role_name* ;

SELECT ROLE ;

GRANT Command (SQL Form)

Once a new role is created, access rights can be added to or withdrawn from the role with GRANT/REVOKE statements.

Roles may be granted privileges on the following database objects.

- Database
- Table
- View
- Macro
- Column
- Triggers
- Stored procedures
- Join and Hash indexes

Roles may not be granted on the following functions (or access rights).

- CREATE ROLE and DROP ROLE
- CREATE PROFILE and DROP PROFILE
- CREATE USER and DROP USER

Exceptions

A role cannot have descendants, i.e., the ALL option of a GRANT/REVOKE statement cannot be applied to a role. The following statement is not allowed.

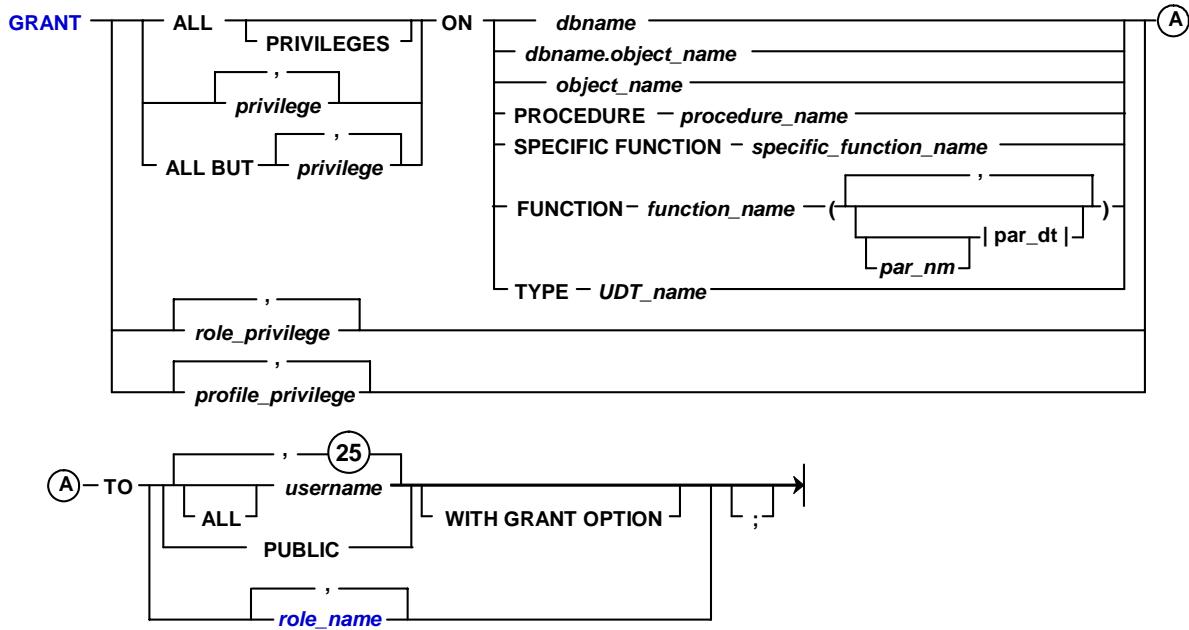
GRANT <right> ON <database object> TO ALL <role name>;

ANSI also disallows a right to be granted to a role with the GRANT option. The following statement is also illegal.

GRANT <right> ON <db object> TO <role name> WITH GRANT OPTION;

GRANT Command (SQL Form)

The GRANT command may be used to grant access rights to roles.



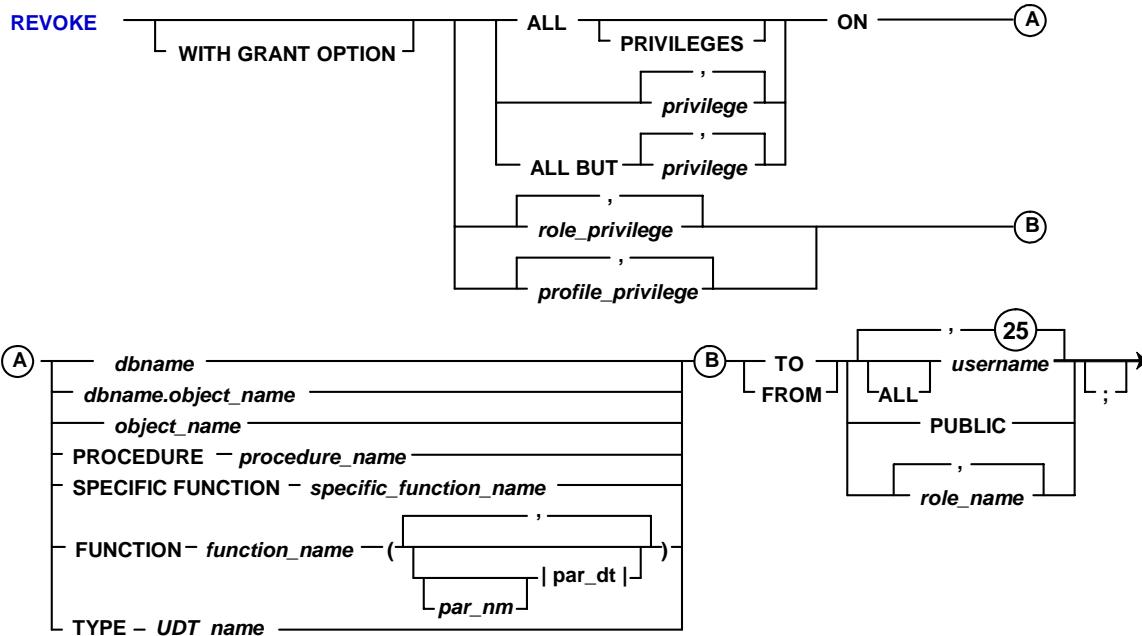
REVOKE Command (SQL Form)

The facing page shows the syntax for the REVOKE Command.



REVOKE Command (SQL Form)

The REVOKE command may be used to revoke access rights from roles.



GRANT and REVOKE Commands (Role Form)

GRANT (Role Format) is used to grant role membership to users or other roles.

role_name

This is a list of one or more comma-separated names of roles to which membership or administrative ability is being granted

TO *user_name* or *role_name*

This is a list of names of role grantees. Grantees can be users or roles; however, a role cannot be granted membership to itself.

WITH ADMIN OPTION

The role grantees have the right to use DROP ROLE, GRANT, and REVOKE statements to administer the roles to which they are becoming members.

A GRANT statement that does not include WITH ADMIN OPTION does not revoke a previously granted WITH ADMIN OPTION privilege from grantee.

REVOKE (Role Format) is used to revoke role membership to users or other roles.

ADMIN OPTION FOR

The role members maintain membership status, but lose the right to use GRANT, REVOKE, and DROP ROLE statements to administer the roles to which they are members.

If ADMIN OPTION FOR does not appear in the REVOKE statement, the system removes the specified roles or users as role members.

role_name

This is a list of one or more comma-separated names of roles from which membership or administrative ability is being revoked. The system ignores duplicate role names.

TO/FROM *user_name* or *role_name*

This identifies the names of role members that are losing membership or administrative ability. Members can be users or roles.



GRANT and REVOKE Commands (Role Form)

The syntax to grant a role to a user (or role) is:

GRANT *role_name* **TO** *user_name* [*role_name*] **WITH ADMIN OPTION** ;

WITH ADMIN OPTION

Gives the role grantee(s) the right to use DROP ROLE, GRANT, and REVOKE statements to administer the roles to which they are becoming members.

The syntax to revoke a role from a user (or role) is:

REVOKE [ADMIN OPTION FOR] *role_name* [, *role_name*] TO *user_name* [, *role_name*] ;

ADMIN OPTION FOR

The role members maintain membership status, but lose the right to administer the roles to which they are members.

If this option is not used, the system removes the specified roles or users as role members.

System Hierarchy (used in following examples)

A system structure for the Teradata database is shown on the facing page and this hierarchy will be used in numerous examples.

Keys to the hierarchy on the facing page are:

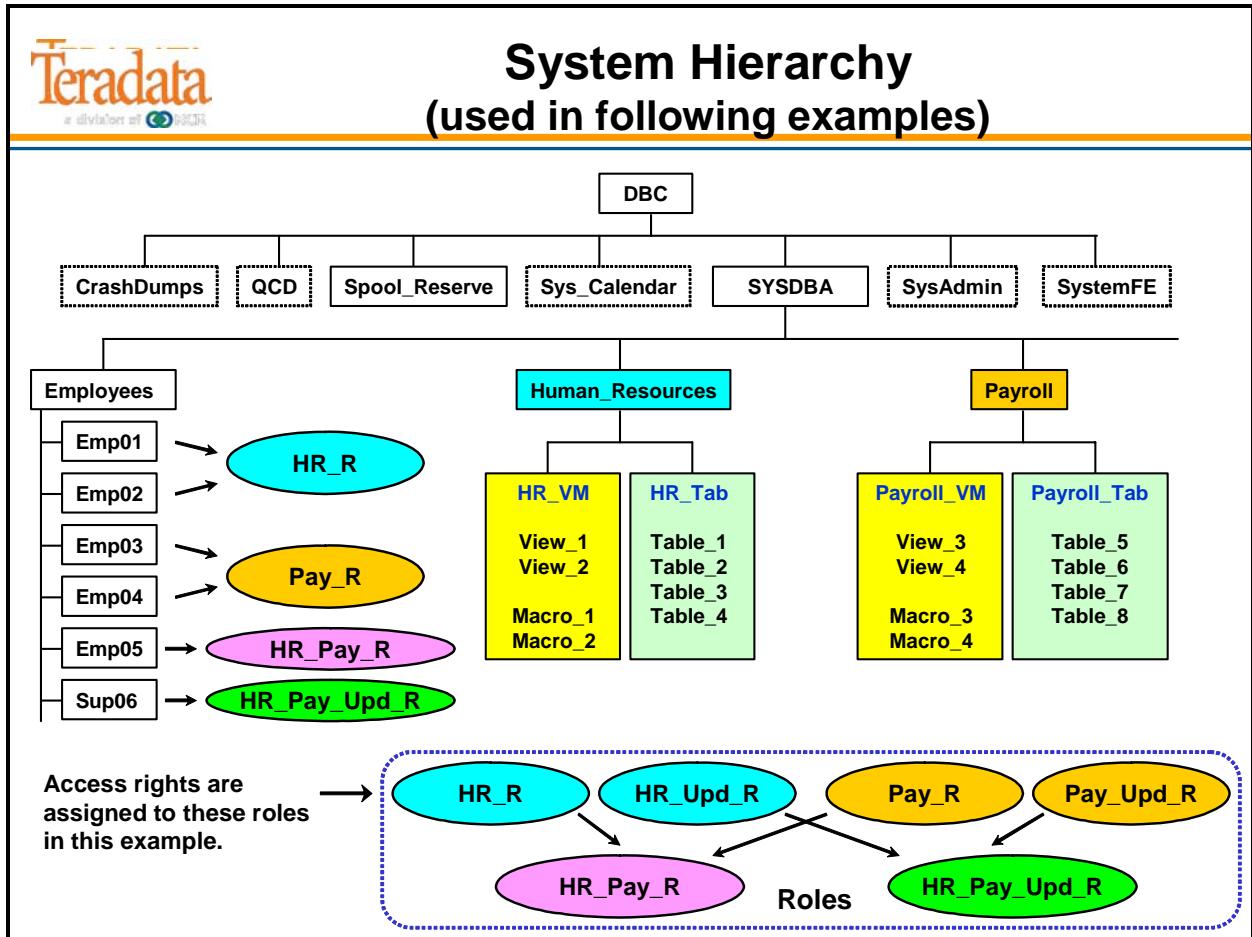
- Roles will have a _R at the end of the role_name. For example, HR_R represents the Human Resources Role.
- Inquiry Users – users that require SELECT and EXECUTE access rights on the views and macros in the VM databases. These users will be assigned either to the HR_R, Pay_R, or the HR_PAY_R.
- Update Users – users that require SELECT, EXECUTE, INSERT, UPDATE, and DELETE access rights on the views and macros in the VM databases. These users will be assigned either to the HR_Upd_R, Pay_Upd_R, or the HR_PAY_Upd_R.

The database HR_VM will have SELECT, EXECUTE, INSERT, UPDATE, and DELETE privileges WITH GRANT OPTION on the database named HR_Tab. Likewise for Payroll_VM and Payroll_Tab.

Dropping a User

When a DROP USER command is issued, both individual rights and role rights granted to the user being dropped will be deleted from the DBC.AccessRights and the DBC.RoleGrants tables. Deletions of database objects within the user space prior to the DROP USER command will cause corresponding deletions of DBC.AccessRights rows for rights granted on these objects to roles and other users/databases.

However, rights granted by the dropped user that are not on objects within its space will remain in the system. This would include role rights. Roles and profiles created by the dropped user will remain in the system.



Example of Using Roles

The facing page contains a simple example of creating a role, assigning access rights to it, and granting the role to users.

The default role for a user will automatically be made the current role for the session when the user first logs on. **The role must be currently granted to the user (otherwise, it is ignored).**

Only a partial listing of the access rights that would be assigned to roles is shown on the facing page. Additionally, these commands would also be executed to complete the example.

- **GRANT SELECT, EXECUTE ON Payroll_VM TO Pay_R;**
- **GRANT SELECT, EXECUTE, INSERT, UPDATE, DELETE ON Payroll_VM TO Pay_Upd_R;**
- **GRANT HR_Upd_R TO HR_Pay_Upd_R; /* nested role */**
- **GRANT Pay_Upd_R TO HR_Pay_Upd_R; /* nested role */**



Example of Using Roles

Create roles.

```
CREATE ROLE HR_R;
CREATE ROLE Pay_R;
CREATE ROLE HR_Pay_R;          CREATE ROLE HR_Upd_R;
                             CREATE ROLE Pay_Upd_R;
                             CREATE ROLE HR_Pay_Upd_R;
```

Assign access rights to the roles (partial listing).

```
GRANT SELECT, EXECUTE           ON HR_VM TO HR_R;
GRANT SELECT, EXECUTE, INSERT, UPDATE, DELETE ON HR_VM TO HR_Upd_R;
```

Grant users permission to use the roles (partial listing).

```
GRANT HR_R          TO Emp01, Emp02;
GRANT Pay_R         TO Emp03, Emp04;
GRANT HR_R          TO HR_Pay_R;    /*nested role*/
GRANT Pay_R         TO HR_Pay_R;    /*nested role*/
GRANT HR_Pay_R      TO Emp05;
GRANT HR_Pay_Upd_R  TO Sup06 WITH ADMIN OPTION;
```

Modify the user to set the default role.

```
MODIFY USER Emp01 AS DEFAULT ROLE = HR_R;
MODIFY USER Emp02 AS DEFAULT ROLE = HR_R;
MODIFY USER Emp03 AS DEFAULT ROLE = Pay_R;
MODIFY USER Emp04 AS DEFAULT ROLE = Pay_R;
MODIFY USER Emp05 AS DEFAULT ROLE = HR_Pay_R;
MODIFY USER Sup06 AS DEFAULT ROLE = HR_Pay_Upd_R;
```

Example of Using Roles (cont.)

The facing page continues the example.

Answer to first question on facing page:

Emp01 does not have UPDATE permission to update the Employee table via the Employee_v view. The error returned is:

5315: The user does not have UPDATE access to HR_VM.Employee_v.Dept_Number.

Answer to second question on facing page:

Both SQL statements work for Emp05 because the access rights for HR_R and Pay_R are nested within HR_Pay_R.



Example of Using Roles (cont.)

Emp01 – is granted to HR_R role and this is the current role.

```
SELECT      *
FROM        HR_VM.Employee_v
ORDER BY    1;                                (success)

UPDATE      HR_VM.Employee_v
SET         Dept_Number=1001
WHERE       Employee_Number=100001;           (fails)
```

Why does this statement fail for Emp01?

Emp05 – is granted to HR_Pay_R role and this is the current role.

```
SELECT      *
FROM        HR_VM.Employee_v
ORDER BY    1;                                (success)

SELECT      *
FROM        Payroll_VM.Salary_v
ORDER BY    1;                                (success)
```

Why do both of these statements succeed for Emp05?

Example of Using Roles (cont.)

The facing page continues the example.

If a user tries to use the SET ROLE command to specify a role they have not been granted access, the user will get the following error:

5621: User has not been granted a specified role.

Answer to first question: The statement fails because Emp05's current role is only provides Select access and this role does not have update permission on Employee_v.

Answer to second question: The statement succeeds because Emp05's current role is now HR_Pay_Upd_R and this role does have update permission on Employee_v.

Answer to third question: Assuming that the default role for Emp05 is HR_Pay_R, the statement will fail until Emp05 uses the SET ROLE command or uses a MODIFY USER command to change the DEFAULT ROLE.

For example:

MODIFY USER Emp05 as DEFAULT ROLE = HR_Pay_Upd_R;



Example of Using Roles (cont.)

Sup06 – is granted to HR_Pay_Upd_R role WITH ADMIN OPTION.

```
GRANT HR_Pay_Upd_R TO Emp05;                                (success)
```

Emp05 – HR_Pay_R is current role.

```
SELECT      *  
FROM        HR_VM.Employee_v  
ORDER BY    1;                                              (success)  
  
UPDATE      HR_VM.Employee_v  
SET         Dept_Number=1001  
WHERE       Employee_Number=100001;                            (fails)
```

Why does this statement fail for Emp05?

Emp05 – executes the following SET ROLE command

```
SET ROLE    HR_Pay_Upd_R;  
UPDATE      HR_VM.Employee_v  
SET         Dept_Number=1001  
WHERE       Employee_Number=100001;
```

Will this UPDATE statement succeed this time?

Will this UPDATE statement succeed the next time Emp05 logs on?

RoleInfo View

The DBC.RoleInfo (or DBC.RoleInfoV) views list all of roles, their creators, and the creation timestamp in the system. This information is taken from the DBC.Roles and the DBC.Dbbase tables.

The DBC.RoleInfoX (or DBC.RoleInfoVX) views only return roles that a user has created. Users that can create roles need the system access right – CREATE ROLE.

Extension to COMMENT command:

COMMENT [ON] ROLE <profile name> [[AS] <comment string>]

- Inserts or retrieves comments in CommentString column of the DBC.Roles table for the named role.

Example:

COMMENT ON ROLE HR_R AS 'SEL and EXE rights for HR_VM';



RoleInfo View

Provides information about roles.

DBC.RoleInfo[V][X]

RoleName	CommentString	CreatorName
CreateTimeStamp	ExtRole	

Example: List role names that exist in the system.

```
SELECT      RoleName, CreatorName, CreateTimeStamp  
FROM        DBC.RoleInfo  
ORDER BY    1;
```

Results:

RoleName	CreatorName	CreateTimeStamp
HR_Pay_R	Sysdba	2008-01-24 17:25:41
HR_Pay_Upd_R	Sysdba	2008-01-24 17:25:44
HR_R	Sysdba	2008-01-24 17:25:02
HR_Upd_R	Sysdba	2008-01-24 17:25:19
Pay_R	Sysdba	2008-01-24 17:25:34
Pay_Upd_R	Sysdba	2008-01-24 17:25:37

RoleMembers View

The DBC.RoleMembers (or DBC.RoleMembersV) views list each role and all of its members.

The DBC.RoleMembersX (or DBC.ROLEMEMBERSVX) views list all roles, if any, directly granted to the user.

For example, Emp05 executes the following statement:

```
SELECT      RoleName, Grantor, WhenGranted, DefaultRole, WithAdmin
FROM        DBC.RoleMembersX
ORDER BY 1;
```

The result is:

<u>RoleName</u>	<u>Grantor</u>	<u>WhenGranted</u>	<u>DefaultRole</u>	<u>WithAdmin</u>
HR_Pay_R	Sysdba	2008-01-24 17:32:51	Y	N



RoleMembers View

Provides information about roles and its members.

DBC.RoleMembers[V][X]

RoleName	Grantee	GranteeKind	Grantor
WhenGranted	DefaultRole	WithAdmin	

Example: List roles and the members that have access to the HR database.

```
SELECT      RoleName, Grantee, GranteeKind, DefaultRole, WithAdmin
FROM        DBC.RoleMembers
WHERE       RoleName IN ('HR_Pay_R', 'HR_Pay_Upd_R', 'HR_R' , 'HR_Upd_R')
ORDER BY    1, 2;
```

Results:

RoleName	Grantee	GranteeKind	DefaultRole	WithAdmin
HR_Pay_R	DBC	User	N	Y
HR_Pay_R	Emp05	User	Y	N
HR_Pay_R	Sysdba	User	N	Y
HR_Pay_Upd_R	DBC	User	N	Y
HR_Pay_Upd_R	Sup06	User	Y	Y
HR_Pay_Upd_R	Sysdba	User	N	Y
HR_R	DBC	User	N	Y
HR_R	Emp01	User	Y	N
HR_R	Emp02	User	Y	N
HR_R	HR_Pay_R	Role	N	N
HR_R	Sysdba	User	N	Y
HR_Upd_R	DBC	User	N	Y
HR_Upd_R	HR_Pay_Upd_R	Role	N	N
HR_Upd_R	Sysdba	User	N	Y

DBC.AccessRights and “Rights” Views

The facing page illustrates the difference between the various “Rights” views of the DBC.AccessRights table.



DBC.AccessRights and “Rights” Views

AllRights[V][X] – lists all rights granted to users in the system.

UserRights[V] – lists all rights granted to the current user.

AllRoleRights[V] – lists all rights granted to roles in the system.

UserRoleRights[V] – lists all rights granted to the enabled roles of the user.

Views

DBC.AllRights and DBC.UserRights
(only select user access rights)

**DBC.AccessRights
(Table)**

User Access Rights

Views

DBC.AllRoleRights and DBC.UserRoleRights
(only select role access rights)

Role Access Rights

AllRoleRights and UserRoleRights Views

The DBC.AllRoleRights[V] and DBC.UserRoleRights[V] views provide information about role and access rights granted to roles in the system.

DBC.UserRoleRights[V] view lists all rights granted to the current role of the user and its nested roles.



AllRoleRights and UserRoleRights Views

AllRoleRights[V] – lists all rights granted to roles in the system.

UserRoleRights[V] – lists all rights granted to the enabled roles of the user.

DBC.AllRoleRights and DBC.UserRoleRights

RoleName	DatabaseName	TableName	ColumnName
AccessRight	GrantorName	CreateTimeStamp	

Example: List current role rights.

```
SELECT      RoleName, DatabaseName, TableName, ColumnName, AccessRight
FROM        DBC.UserRoleRights
ORDER BY    1;
```

Example Results for Emp01:

RoleName	DatabaseName	TableName	ColumnName	AccessRight
HR_R	HR_VM	All	All	R
HR_R	HR_VM	All	All	E

Example Results for Emp05:

The default role of Emp05 is HR_Pay_R which has 2 nested roles.

HR_R and Pay_R

RoleName	DatabaseName	TableName	ColumnName	AccessRight
HR_R	HR_VM	All	All	R
HR_R	HR_VM	All	All	E
Pay_R	Payroll_VM	All	All	R
Pay_R	Payroll_VM	All	All	E

Steps to Implementing Roles

The facing page identifies a sequence of steps to consider when implementing roles. A sample query and results are also provided.

GRANT PUBLIC Implementation Change

The PUBLIC option of the GRANT command allows privileges to be granted to all existing and future users.

Starting with V2R5, the PUBLIC implementation (also works with the ALL DBC syntax) was changed from one dictionary row per PUBLIC right per user to one row per right. That is, a single row per access right is placed in the DBC.AccessRights table when the PUBLIC option is used.

The facing page shows the use of PUBLIC. The following example shows that ALL DBC effectively works the same as PUBLIC.

SELECT COUNT(*) FROM DBC.AllRights;

Result: Count(*)
4447

GRANT SELECT ON HR_VM.View6 TO ALL DBC;

SELECT COUNT(*) FROM DBC.AllRights;

Result: Count(*)
4448 (only one access right is added)



Steps to Implementing Roles

1. Identify individual rights to be converted into role rights.
2. Create roles and grant appropriate rights to each role.
3. Grant roles to users and assign users their default roles.
4. Revoke from users individual rights that have been replaced by role rights.

Sample query to identify individual rights that may be good candidates for conversion to roles.

```

SELECT      DatabaseName,
            TVMName,
            COUNT(*)          AS RightsCount
FROM        DBC.AccessRights AR,
            DBC.TVM          TVM,
            DBC.DBase         DBASE
WHERE       AR.tvmid = TVM.tvmid
AND         AR.databaseid = DBASE.databaseid
AND         AR.fieldid = 0
GROUP BY    DatabaseName, TVMName
ORDER BY    3 DESC;
  
```

Results:

DatabaseName	TableName	RightsCount
DS	All	110
QCD	All	86
HR_Tab	All	72
HR_VM	All	68
Payroll_VM	All	67
Payroll_Tab	All	67

Profiles

Profiles define system parameters. Assigning a profile to a group of users makes sure that all group members operate with a common set of parameters.

To manage system parameters for groups, a database administrator can:

- Create a different profile for each user group, based on system parameters that group members share.

You can define values for all or a subset of the parameters in a profile. If you do not set the value of a parameter, the system uses the setting defined for the user in a CREATE USER or MODIFY USER statement.

- Assign profiles to users.

The parameter settings in a user profile override the settings for the user in a CREATE USER or MODIFY USER statement.

Like roles, the concept of ownership and ownership hierarchy is not be applicable to profiles.



Profiles

What is a “profile”?

- A profile is a set of common user parameters that can be applied to a group of users.
- Profile parameters include:
 - Account id(s)
 - Default database
 - Spool space allocation
 - Temporary space allocation
 - Password attributes (expiration, etc.)

What are the advantages of using “profiles”?

- Profiles simplify user management.
 - A change of a common parameter requires an update of a profile instead of each individual user affected by the change.

How are “profiles” managed?

- New DDL commands, tables, view, command options, and access rights.
 - CREATE PROFILE, MODIFY PROFILE, DROP PROFILE, and SELECT PROFILE
 - New system table - DBC.Profiles
 - New system views - DBC.ProfileInfo[V][X]

Example of Simplifying User Management

The profile concept provides a solution to the following problem.

A customer has a group of 10,000 users that are assigned the same amount of spool space, the same default database, and the same account ID. Changing any of these parameters for 10,000 users is a very time-consuming task for the database administrators.

The database administrators' task will be simplified if they can create a profile that contains one or more system parameters such as accounts ids, default database, spool space and temporary space. This profile is assigned to the group of users.

This would simplify system administration because a parameter change requires updating only the profile instead of each individual user.

In summary, a set of parameters may be assigned certain values in a profile and this profile may be assigned to a group of users and thereby have them share the same settings. This makes changing parameters for a group of users a single step instead of a multi-step (one for each user in the group) process.



Example of Simplifying User Management

Example:

- **The problem:**

- A customer has a group of 10,000 users that are assigned the same spool space, the same default database, and the same account ID.
- Changing any of these parameters for 10,000 users can be a time-consuming task.

- **A solution using profiles:**

- Create a profile that contains these parameters and assign that profile to the users.
- This would simplify system administration because a parameter change requires updating only the profile instead of each individual user.

Implementing Profiles

The CREATE PROFILE and DROP PROFILE access rights are system rights. These rights are not on a specific database object. Note that the PROFILE privileges can only be granted to a user and not to a role or database.

Profiles enable you to manage the following common parameters:

- Account strings, including ASE codes and Performance Groups
- Default database
- Spool space
- Temporary space
- Password attributes, including:
 - Expiration
 - Composition (length, digits, and special characters)
 - Allowable logon attempts
 - Duration of user lockout (indefinite or elapsed time)
 - Reuse of passwords

Note: In the example on the facing page, another technique of granting CREATE PROFILE and DROP PROFILE to Sysdba is to use the following SQL.

GRANT PROFILE TO SYSDBA WITH GRANT OPTION;

The key word PROFILE will give both the CREATE PROFILE and DROP PROFILE access rights.



Implementing Profiles

What access rights are used to support profiles?

- CREATE PROFILE – needed to create new profiles
- DROP PROFILE – needed to modify and drop profiles

Who is allowed to create and modify profiles?

- Initially, only DBC has the CREATE PROFILE and DROP PROFILE access rights.
- As DBC, give the “profile” access rights to the database administrators (e.g. Sysdba).

GRANT CREATE PROFILE, DROP PROFILE TO Sysdba WITH GRANT OPTION;

How are users associated with a profile?

- The CREATE PROFILE command is used to create a profile of desired attributes.
CREATE PROFILE Employee_P AS ... ;
- The PROFILE option (new) is used with CREATE USER and MODIFY USER commands to assign a user to a specific profile.
CREATE USER Emp01 AS ..., PROFILE = Employee_P;
MODIFY USER Emp02 AS PROFILE = Employee_P;

Impact of Profiles on Users

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

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 TEMP space allocations are imposed immediately. This will affect 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.
- Account IDs and a default database are considered at next logon unless the member submits a SET SESSION ACCOUNT statement, in which case the account ID must agree with the assigned profile definition.

Order of Precedence

With profiles, there are 3 ways of setting accounts and default database. The order of precedence (from high to low) is as follows:

1. The DATABASE statement is used to set the current default database or the SET SESSION ACCOUNT is used to set the account ID. However, a user can only specify a valid account ID.
2. Specify them in a profile and assign the profile to a user.
3. Specify accounts or default database for a user through the CREATE USER/MODIFY USER statements.



Impact of Profiles on Users

The assignment of a profile to a group of users is a way of ensuring that all members of a group operate with a common set of parameters.

Profile definitions apply to every assigned user, overriding specifications at the system or user level.

- However, any profile definition can be NULL or NONE.

All members inherit changed profile parameters. The impact on current users is as follows:

- SPOOL and TEMPORARY space allocations are imposed immediately.
- Password attributes take effect upon next logon.
- Database and Account IDs are considered at next logon unless the member submits a SET SESSION ACCOUNT statement.

Order of Precedence for parameters:

1. Specify database or account ID at session level
2. Specified parameters in a Profile
3. CREATE USER or MODIFY USER statements

CREATE/MODIFY PROFILE Statement

The CREATE PROFILE statement enables you to add new profiles to the system. The CREATE PROFILE access right is required in order to execute this command. The syntax is shown on the facing page.

Profile names come from their own name space. Like roles, the concept of ownership and ownership hierarchy is not applicable to profiles.

A parameter not set in a profile will have a value of NULL. Resetting a parameter to NULL will cause the system to apply the user's setting instead. In a profile, the SPOOL and TEMPORARY limits may not exceed the current space limits of the user submitting the CREATE/MODIFY PROFILE statement.

The default database specified in a profile need not refer to an existing database. This is consistent with current CREATE USER and MODIFY USER statements where a non-existent default database may be specified. An error will be returned when the user tries to create an object within the non-existent database.

It is not necessary to define all of the parameters in a profile, a subset will also do. The parameter values in a user profile take precedence over the values set for the user. For example, if a user is assigned a profile containing Default Database and Spool Space, the profile settings will override the individual settings previously made via a CREATE USER or MODIFY USER statement.

Accounts in a profile will also override, not supplement, any other accounts the user may have. The assignment of a profile to a group of users is a way of ensuring that all group members operate with a common set of parameters. The first account in a list will be the default account.

If a parameter in a profile is not set, then the user's setting will be applied.

Note when using the CREATE USER command:

- When creating a new user, if the PROFILE option specifies a Profile that does not exist, you will get the following error.

Error 5653: Profile 'profile_name' does not exist.

Modify Profile Statement

The MODIFY PROFILE statement enables you to change the options of an existing profile. The DROP PROFILE access right is required in order to execute this command. The syntax is similar to CREATE PROFILE and is also shown on the facing page.

To remove a profile from a user,

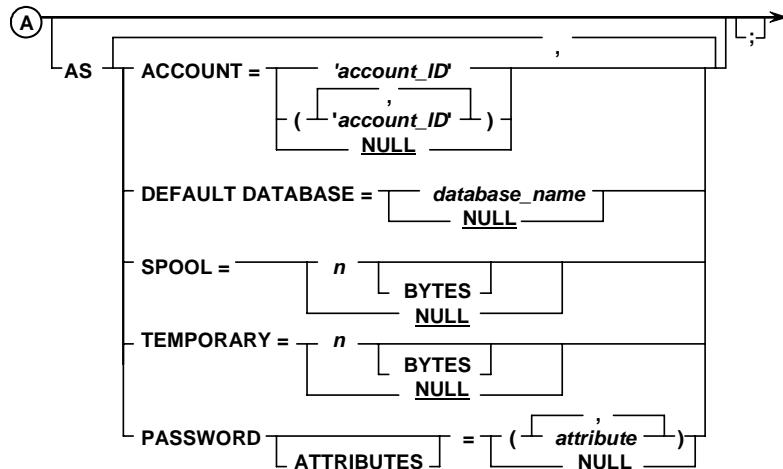
MODIFY USER *username* AS PROFILE = NULL;



CREATE/MODIFY PROFILE Statement

CREATE PROFILE *profile_name* _____ A

MODIFY PROFILE *profile_name* _____ A



Password Attributes (CREATE/MODIFY PROFILE)

The facing page describes the Password Attributes associated with the CREATE PROFILE and MODIFY PROFILE commands.

If a parameter is not specified in a profile and is not specified with the CREATE USER or MODIFY USER statement, the following list of defaults applies.

Parameter	Default Value
Account ID	The default account ID (first account ID of the immediate owner of the user).
Performance Group	\$M
DEFAULT DATABASE	Username
SPOOL	The same SPOOL value as the owner of the space in which the user is being created.
TEMPORARY	The same TEMPORARY value as the owner of the space in which the user is being created.

If the password attributes option is not specified in a profile, then the password attributes specified in the DBC.SysSecDefaults table are used for the users assigned to this profile.



Password Attributes (CREATE/MODIFY PROFILE)

Password Attributes

EXPIRE	=	\boxed{n}	(# of days; 0 doesn't expire)
MINCHAR	=	\boxed{n}	(range is 1 - 30)
MAXCHAR	=	\boxed{n}	(range is 1 - 30)
DIGITS	=	\boxed{c}	(options are Y, y, N, n, R, r)
RESTRICTWORDS	=	\boxed{c}	(options are Y, y, N, n) (This is a 12.0 feature)
SPECCHAR	=	\boxed{c}	(options are Y, y, N, n, ...)
MAXLOGONATTEMPTS	=	\boxed{n}	(# of attempts; 0 = never locked)
LOCKEDUSEREXPIRE	=	\boxed{n}	(# of minutes; 0 = not locked; -1 = locked indefinitely)
REUSE	=	\boxed{n}	(# of days; 0 - reuse immediately)

Teradata Password Control

Forcing users to create passwords with one or more of the special character options enhances password security. It also may make the password harder for the user to remember and to type in at logon. Consider these two factors when deciding how elaborate the password special character requirements should be for your system

Many passwords would be relatively easy for an intruder to guess, especially if some of the letters are known. Forcing users to create passwords with one or more digits enhances password security.

When specifying the maximum password length, keep in mind that some users may try to create a password of maximum length. Because it is more difficult to remember a long password, the user is more likely to write it down rather than memorize it – and it is strongly recommended that users do not write passwords down.

Adding and Removing Restricted Words

The PasswordRestrictWords (DBC.SysSecDefaults) or RestrictWords (profiles) parameter determines whether or not a password is subject to the content restrictions defined in the DBC.PasswordRestrictions list. A default set of Restricted Words is automatically installed when a system is upgraded to Teradata Database 12.0 or greater.

You can add words to the Restricted Words list, using the following form:

```
INSERT INTO DBC.PasswordRestrictions  
VALUES ('newrestrictedword');
```

Note: Although the default Restricted Words list is composed of English words, the words you add can be in any supported character set.

You can also remove words from the list using the following form:

```
DELETE FROM DBC.PasswordRestrictions WHERE  
(RestrictedWords = 'wordtobedeleted');
```

The DBC.RestrictedWords (or DBC.RestrictedWordsV) views can be used to view restricted words.



Teradata Password Control

Description

- At the system level, the DBC.SysSecDefaults table has password parameters to control system-wide password attributes. Similar parameters are available at the profile level to establish password attributes for a group of users.
- Starting with **Teradata V2R6.1**, the DBS Password Control feature provides additional requirements on valid Teradata user passwords.
 - These requirements, which mainly consist of enforcing character variation within a password string, can be enabled or disabled by the DBA/Security Administrator on a system/user basis.
- Starting with **Teradata 12.0**, you can specify whether or not a password is subject to the content restrictions defined in the table DBC.PasswordRestrictions.

Customer Benefit

- These new features allow for the requirement and enforcement of stronger passwords.
- Many passwords would be relatively easy for an intruder to guess, especially if they contain common words or names. Forcing users to create passwords that do not use common words or names enhances password security.

Teradata Password Control (cont.)

The PasswordDigits or Digits parameter determines if digits may be used in a password.

The default value for the PasswordDigits or the Digits parameter is **Y**: Digits are allowed in a password.

The acceptable values for the PasswordDigits (DBC.SysSecDefaults – table or DBC.SecurityDefaults - view) or Digits (Profile) parameter are:

- **Y** = Digits are allowed
- **N** = Digits are not allowed
- **R** = At least one digit is required

Note: The values are not case sensitive.

Password Special Characters

One of the key password parameters is "PasswordSpecChar" or "SpecChar". This parameter determines how ASCII special characters can be used in a password. It includes the following options:

- special characters are allowed/not allowed/required
- passwords must contain at least one alpha character
- no password can contain the database username
- passwords must contain a mixture of upper/lower case letters

The default value of this parameter is **Y**:

- special characters are allowed in a password
- username is allowed in the password string
- alpha characters are allowed but not required
- mixed upper and lower case characters are allowed but not required



Teradata Password Control (cont.)

Options

- The **PasswordSpecChar** (or **SpecChar**) parameter is used to establish the following password control rules.
 - Do not allow a password to contain the user name
 - Allow or require a mixture of upper/lower case characters
 - Allow or require at least one alpha character
 - Allow, not allow, or require at least one special character
- The **PasswordDigits** (or **Digits**) parameter is used to allow, not allow, or require a numeric digit in the password.
- The **PasswordRestrictWords** (or **RestrictWords**) parameter is used to indicate that a password cannot contain one of the "restricted words".
- Note: The **DBC.SecurityDefaults** (view) can be used to view/update **DBC.SysSecDefaults**.

Teradata Password Control Options

The PasswordSpecChar parameter (DBC.SysSecDefaults) or SPECCHAR (Profile) determines how special characters can be used in a password. It includes the following options:

- special characters are allowed/not allowed /required
- passwords must contain at least one alpha character
- no password can contain the database username
- passwords must contain a mixture of upper/lower case letters

The default value of the PasswordSpecChar parameter is **Y**:

- special characters are allowed in a password
- username is allowed in the password string
- alpha characters are allowed but not required
- mixed upper and lower case characters are allowed but not required

A Password *can* contain ...

- 1 to 30 characters (UTF-8 *or* UTF-16 characters)
- Letters **A** through **Z** and/or **a** through **z**
- Digits **0** through **9** in single-byte or multi-byte form
- **Note:** A password can be all-numeric only if it is enclosed in quotes as shown in the following example: password = “12341234”
- The following special characters, in either single-byte or multi-byte form:
- \$ (dollar sign)
- _ (underscore)
- # (pound sign)
- Other special characters may be used (if they are not specifically prohibited by the rules in the reference manuals) *and* if the password is enclosed in quotes (“ ”).



Teradata Password Control Options

Options Table for PasswordSpecChar and SpecChar

Option PasswordSpecChar or SpecChar	N	Y	A	B	C	D	E	F	G	H	I	J	K	L	M	O	P	R
Rule Username	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
Rule Upper/Lower	Y	Y	Y	Y	Y	Y	R	R	R	Y	Y	Y	Y	Y	R	R	R	
Rule One Alpha	Y	Y	Y	R	R	R	R	R	Y	Y	Y	R	R	R	R	R	R	
Rule Special Chars.	N	Y	R	N	Y	R	N	Y	R	N	Y	R	N	Y	R	N	Y	R

N – Not Allowed, Y – Allowed, but not required, R – Required

Note: These options are not case sensitive. You can use "A" or "a".

CREATE PROFILE Example

The facing page contains a simple example of creating a profile, assigning it to a user, and then removing it from the user with the MODIFY USER command.

As mentioned previously, profile definitions apply to every assigned user, overriding specifications at the system or user level.

Answers to first two questions on facing page: 500e6 and \$M_&S_&D&H

Answers to second two questions on facing page: 200e6 and \$M



CREATE PROFILE Example

Create a profile.

```
CREATE PROFILE Employee_P AS
  ACCOUNT      = ('$M_&S_&D&H', '$L_&S_&D&H')
  DEFAULT DATABASE = HR_VM,
  SPOOL        = 500e6,
  TEMPORARY    = 200e6,
  PASSWORD = (EXPIRE = 90, MINCHAR = 8, MAXLOGONATTEMPTS = 3,
               LOCKEDUSEREXPIRE = 60, REUSE = 180,
               DIGITS = 'R', RESTRICTWORDS = 'Y', SPECCHAR = 'P');
```

Assign the profile to a user.

```
CREATE USER Emp01 AS PERM = 0,
          PASSWORD = emp01pass,
          PROFILE = Employee_P,
          SPOOL = 200e6,
          ACCOUNT = '$M';
```

What is the spool space limit for Emp01?

What is the default account code for Emp01?

Assume this command is executed: **MODIFY USER Emp01 AS PROFILE = NULL;**

What is the spool space limit for Emp01?

What is the default account code for Emp01?

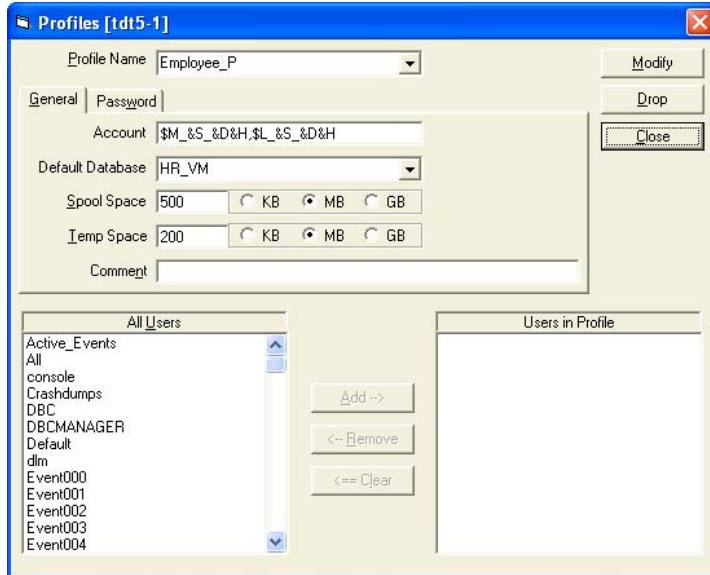
Teradata Administrator CREATE PROFILE Example

The facing page contains an example of creating a profile using Teradata Administrator.



Teradata Administrator CREATE PROFILE Example

From Teradata Administrator, use the Tools > Administer Profiles menus.



These 2 options are equivalent to:

**SPOOL=500e6,
TEMPORARY=200e6**

```
CREATE PROFILE Employee_P AS
  ACCOUNT = ('$M_&S_&D&H', '$L_&S_&D&H')
  DEFAULT DATABASE = HR_VM, SPOOL = 500e6, TEMPORARY = 200e6, ...
```

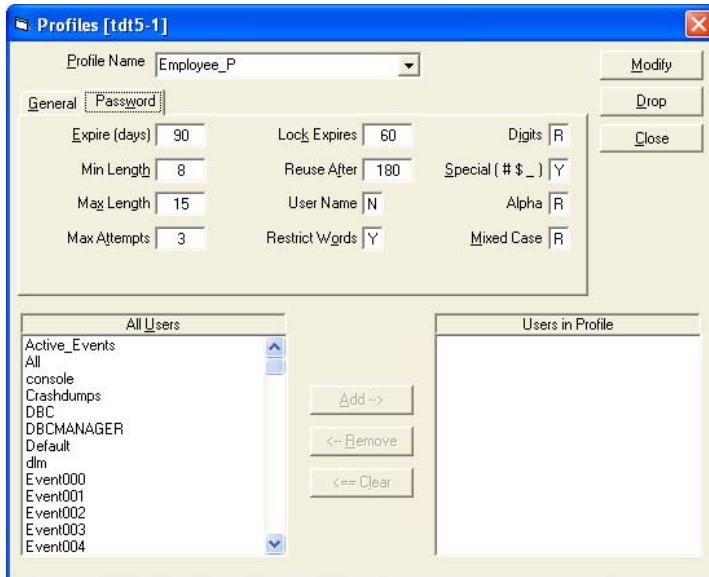
CREATE PROFILE Example (cont.)

The facing page continues the example of creating a profile using Teradata Administrator.



Teradata Administrator CREATE PROFILE Example (cont.)

From Teradata Administrator, use the Tools > Administer Profiles menus.



These options are equivalent to:

**DIGITS='R',
RESTRICTWORDS='Y',
SPECCHAR='P'**

CREATE PROFILE Employee_P ...

```
PASSWORD = (EXPIRE=90, MINCHAR=8, MAXCHAR=15, MAXLOGONATTEMPTS=3,
LOCKEDUSEREXPIRE=60, REUSE=180, DIGITS='R', RESTRICTWORDS='Y', SPECCHAR='P');
```

DROP PROFILE Statement

The DROP PROFILE statement drops the named profile. The DROP PROFILE access right is required in order to execute this command.

The syntax is simply:

DROP PROFILE *profile_name*;

When a profile is dropped, users who have the profile assigned to them continue to have that profile assigned to them; the system does *not* reset the profile for the affected users to NULL. Affected users receive no warnings or errors the next time they log on.

The effects of re-creating a profile with the same name as the dropped profile are not immediate. The parameter settings in the re-created profile take effect the next time that users (who are assigned the profile) log on.

DROP PROFILE has the following effects on users (sessions) logged on with the profile:

- Spool and temporary space settings immediately change to the settings defined for the affected users.
- Account and database settings change to the settings defined for the affected users the next time the users log on or explicitly change the settings.

However, changes to the list of valid account IDs take effect immediately. Users may only explicitly change to an account ID in the list of account IDs available to them.



DROP PROFILE Statement

Syntax:

```
DROP PROFILE profile_name;
```

Notes:

- If a profile is dropped, users who have the profile assigned to them continue to have that profile assigned to them.
- Effects on sessions logged on with the profile that has been dropped:
 - Spool and temporary space settings immediately change user's settings.
 - Account and database settings change to user's settings on next log on or if user explicitly changes the settings.
 - Users may only explicitly change to an account in the list of account IDs available to them.
- The effects of re-creating a profile with the same name as the dropped profile are not immediate.
 - The parameter settings in the re-created profile take effect the next time the users log on.

ProfileInfo View

The DBC.ProfileInfo view will list all profiles and their parameter settings. This information is taken from the DBC.Profiles table.

The DBC.ProfileInfoX view will list the profile, if any, and its parameter settings for the current user.

Extension to COMMENT command:

COMMENT [ON] PROFILE <profile name> [[AS] <comment string>]

- inserts or retrieves comments in CommentString column of the DBC.Profiles table for the named profile.

Implementation Note:

Like accounts for the DBC.Dbase table, only the default account is stored in the DBC.Profiles table. All other accounts associated with the profile will be stored in DBC.Accounts table. The ProfileInfo view provides the first or default account ID.



ProfileInfo View

Provides information about profiles that exist in the system.

DBC.ProfileInfo[V][X]

ProfileName	DefaultAccount	DefaultDB
SpoolSpace	TempSpace	ExpirePassword
PasswordMinChar	PasswordMaxChar	PasswordDigits
PasswordSpecChar	PasswordRestrictWords	MaxLogonAttempts
LockedUserExpire	PasswordReuse	CommentString
CreatorName	CreateTimeStamp	LastAlterName
LastAlterTimeStamp		

Example:

List profiles that exist
in the system.

```
SELECT ProfileName
      ,DefaultAccount AS "Def Acct"
      ,DefaultDB
      ,SpoolSpace
      ,TempSpace
   FROM DBC.ProfileInfo
  ORDER BY 1;
```

Example Results:

ProfileName	Def Acct	DefaultDB	SpoolSpace	TempSpace
Cust_Service_P	\$M_&D&H	CS_VM	200000000	100000000
Cust_Service_Gold_P	\$H_&D&H	CS_VM	200000000	100000000
Employee_P	\$M_&S_&D&H	HR_VM	500000000	200000000
Payroll_P	\$M_&D&H	Payroll_VM	200000000	100000000

Miscellaneous SQL Functions

The facing page contains 3 simple examples of SQL functions that a user can execute to identify their profile information.

As you may notice, the DBC.AccountInfo view has a new column (UserOrProfile). This corresponds to a new column (RowType) that has been added to the DBC.Accounts table.

A new column, RowType, has been added to the DBC.Accounts table. This is necessary because profile and user names come from separate name spaces. If profile accounts are also stored in DBC.Accounts, they will be confused with accounts of a user with the same name. Hence, the RowType column is necessary to distinguish a user account from a profile account. This column will have a value of P (for Profile) or U (for User).



Miscellaneous SQL Functions

Example 1: As Emp01, identify the current user, role, profile, and database information.

```
SELECT USER, ROLE, PROFILE, DATABASE;
```

Result 1:

User	Role	Profile	Database
EMP01	HR_R	EMPLOYEE_P	HR_VM

Example 2: As Emp01, list the profile attributes.

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

Result 2:

ProfileName	DefaultAccount	DefaultDB	SpoolSpace	TempSpace
Employee_P	\$M_&S_&D&H	HR_VM	500000000	200000000

Example 3: As Emp01, list account information.

```
SELECT * FROM DBC.AccountInfoX;
```

Result 3:

Name	AccountName	UserOrProfile
Employee_P	\$M_&S_&D&H	Profile
Employee_P	\$L_&S_&D&H	Profile
Emp01	\$M	User

Summary

The facing page summarizes some important concepts regarding this module.

A summary of the rules for using roles are as follows:

- You can grant one or more roles to one or more users and/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.
- An access privilege granted to an existing role immediately affects any user and/or role that is specified as a recipient in the GRANT statement and currently active within a session.
- The privileges of a role granted to another role are inherited by every user member of the grantee role.
- When a user logs on, the assigned default role is the initial current role for the session and is used to authorize access after all checks against individually granted rights have failed.
- Once the session is active, the user can submit a SET ROLE statement to change or nullify the current role.



Summary

- A role is simply a collection of access rights.
- Rights are first granted to a role and the role is then granted to users.
 - `CREATE ROLE role_name;`
 - `GRANT access_right ON object TO role_name;`
 - `GRANT role_name TO user_name;`
- A profile is a set of common user parameters that can be applied to a group of users.
- The CREATE PROFILE command is used to create a profile of desired attributes.
 - `CREATE PROFILE profile_name AS ... ;`
- The PROFILE option (new) is used with CREATE USER and MODIFY USER commands to assign a user to a specific profile.
 - `CREATE USER user1 AS ..., PROFILE = prof_name;`
 - `MODIFY USER user2 AS PROFILE = prof_name;`

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

Answer the following questions:

1. List 3 advantages of utilizing roles and profiles.

2. How many levels of role nesting are currently allowed? _____
3. True or False. A user can use the SET ROLE command to set their current role to any defined role in the system.
4. True or False. Roles may only be granted to users and other roles.

Match each term to the definition.

- | | |
|--|---|
| <input type="checkbox"/> 1. WITH ADMIN OPTION | A. Established by the SET ROLE command |
| <input type="checkbox"/> 2. CREATE ROLE | B. Lists the roles currently available to a user |
| <input type="checkbox"/> 3. DBC.RoleInfo | C. System access right needed to create a role |
| <input type="checkbox"/> 4. DBC.UserRoleRights | D. Lists all of a user's role rights - including nested roles |
| <input type="checkbox"/> 5. DEFAULT ROLE | E. Allows the user to assign other users to the role |
| <input type="checkbox"/> 6. Current role | F. Option with the MODIFY USER statement |

Lab Exercise 46-1

The following page continues this lab exercise.



Lab Exercises

Lab Exercise 46-1

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to work with access rights. This lab will also provide an opportunity to use some system views.

Tasks

1. Using the DBC.AllRights view, find the total number of rows in the DBC.AccessRights table assigned to users.

Total row count of user rights (AllRights view) _____

Using the DBC.AllRoleRights view, find the total number of rows in the DBC.AccessRights table assigned to roles.

Total row count of role rights (AllRoleRights view) _____

2. Using the DBC.UserRights view, take a look at the databases and tables on which you currently hold rights.

Total number of rows returned _____

How do you think most of these access rights were granted? _____

Execute the following SQL command and then recheck the number of Access Rights you have.

`CREATE TABLE Emp_Phone2 AS PD.Emp_Phone WITH NO DATA;`

Total number of user rights returned _____

How many new access rights were created? _____

Lab Exercise 46-1 (cont.)

The following page continues this lab exercise.



Lab Exercises

Lab Exercise 46-1 (cont.)

Tasks

3. For your Emp_Phone2 table, use the GRANT command to give the SELECT access right to the user named AP.

Use the GRANT command to give the SELECT WITH GRANT access right to the user PD.

Check the total number of user rights returned _____

Did this count change? _____

If not, why not? _____

Use the DBC.UserGrantedRights view to show any user rights that you may have explicitly granted.
(Qualify this view with a WHERE clause and specify the Grantee as AP or PD.)

Lab Exercise 46-2

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.RoleInfo[V][X]

RoleName	CommentString	CreatorName
CreateTimeStamp	ExtRole	



Lab Exercises

Lab Exercise 46-2

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to use profiles and roles. This lab will also provide an opportunity to use some system views.

What you need

Tables from PPI exercise and a user account with system role and profile privileges.

Tasks

1. Three roles with admin option are available for your use. The role names have your user name incorporated into them. Your role names will be unique in the system. For example, if your user name is "tljc02", then your role names are:

Role1_tljc02 "Role1" (note - this role will be referred to as "Role1" throughout this lab).
Role2_tljc02 "Role2"
Role3_tljc02 "Role3"

Use the DBC.RoleInfo and DBC.RoleInfoX views to display information about roles in the system.

What is the primary difference between using these 2 views?

Using the DBC.RoleInfo view, find the total number of roles defined in the system.

Using this view, how many roles are defined in this system? _____

Lab Exercise 46-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.ProfileInfo[V][X]

ProfileName	DefaultAccount	DefaultDB
SpoolSpace	TempSpace	ExpirePassword
PasswordMinChar	PasswordMaxChar	PasswordDigits
PasswordSpecChar	PasswordRestrictWords	MaxLogonAttempts
LockedUserExpire	PasswordReuse	CommentString
CreatorName	CreateTimeStamp	LastAlterName
LastAlterTimeStamp		



Lab Exercises

Lab Exercise 46-2 (cont.)

2. Grant the following access rights to the specified roles as follows:

<u>Access Rights</u>	<u>Tables</u>	<u>Role Name</u>
SELECT	Orders, Orders_2008	"Role1"
SELECT	Orders_PPI_M	"Role2"
INSERT, UPDATE, DELETE	Orders_PPI_M	"Role3"

3. Create a user profile with a profile name that is the same as your user name (e.g., "tljcx" where xx is your student number). The attributes of your profile are:

ACCOUNT	= '\$M',
DEFAULT DATABASE	= your database (e.g., tljcx)
SPOOL	= 50e6,
TEMPORARY	= 50e6,
PASSWORD	= (EXPIRE = 91, MINCHAR = 6, MAXLOGONATTEMPTS = 4, LOCKEDUSEREXPIRE = 1, REUSE = 365);

4. Use the DBC.ProfileInfo view to display information about profiles in the system.

Lab Exercise 46-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.ProfileInfo[V][X]

ProfileName	DefaultAccount	DefaultDB
SpoolSpace	TempSpace	ExpirePassword
PasswordMinChar	PasswordMaxChar	PasswordDigits
PasswordSpecChar	PasswordRestrictWords	MaxLogonAttempts
LockedUserExpire	PasswordReuse	CommentString
CreatorName	CreateTimeStamp	LastAlterName
LastAlterTimeStamp		



Lab Exercises

Lab Exercise 46-2 (cont.)

5. Create two new users in the system with the following attributes.

User name: tljcx_A (where xx is your student number) - referred to as "User_A"

Perm space: 0

Password: tljcx_A

Profile: tljcx

Default Role: Role1_tljcx

User name: tljcx_B (where xx is your student number) - referred to as "User_B"

Perm space: 0

Password: tljcx_B

Profile: tljcx

Default Role: Role2_tljcx

6. Grant "Role1" to "User_A" (or tljcx_A).

Grant "Role2" to "User_B" (or tljcx_B).

Grant "Role2" to "Role3".

Grant "Role3" to "User_B" (or tljcx_B).

7. Logon to Teradata as "User_A" (or tljcx_A).

Were you prompted to enter a new password? If so, set the password to a new value.

Why were you prompted to enter a new password? _____

Lab Exercise 46-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.UserRoleRights[V]

RoleName	DatabaseName	TableName
ColumnName	AccessRight	GrantorName
CreateTimeStamp		

DBC.RoleMembers[V][X]

RoleName	Grantee	GranteeKind
Grantor	WhenGranted	DefaultRole
WithAdmin		



Lab Exercises

Lab Exercise 46-2 (cont.)

8. As "User_A", execute the following SQL statements and indicate if SELECT is allowed or not.

SELECT COUNT(*) FROM Orders;
SELECT COUNT(*) FROM Orders_PPI_M;

Permitted or not? _____
Permitted or not? _____

9. As "User_A", use the DBC.RoleMembersX and DBC.UserRoleRights views to view the current role of the user, any nested roles, and access rights for the roles.

Are there any other roles that "User_A" has available? _____

Are there any nested role rights? _____

How many user role rights are available to "User_A"? _____

OPTIONAL

10. Logon to Teradata as "User_B".

If prompted, set the password to a new value.

Lab Exercise 46-2 (cont.)

Check your understanding of the concepts discussed in this module by completing the lab exercises as directed by your instructor.

DBC.UserRoleRights[V]

RoleName	DatabaseName	TableName
ColumnName	AccessRight	GrantorName
CreateTimeStamp		

DBC.RoleMembers[V][X]

RoleName	Grantee	GranteeKind
Grantor	WhenGranted	DefaultRole
WithAdmin		



Lab Exercises

Lab Exercise 46-2 (cont.)

OPTIONAL

11. As "User_B", execute the following SQL statements and indicate if SELECT is allowed or not.

SELECT COUNT(*) FROM Orders;
SELECT COUNT(*) FROM Orders_PPI_M;
DELETE Orders_PPI_M;

Permitted or not? _____
Permitted or not? _____
Permitted or not? _____

12. As "User_B", use the DBC.RoleMembersX and DBC.UserRoleRights views to view the current role of the user, any nested roles, and access rights for the roles.

Are there any other roles that "User_B" has available? _____
Are there any nested role rights? _____
How many user role rights are available to "User_B"? _____

13. As "User_B", use the SET ROLE command to set the current role to "Role3".

SELECT COUNT(*) FROM Orders;
SELECT COUNT(*) FROM Orders_PPI_M;
DELETE Orders_PPI_M;

Permitted or not? _____
Permitted or not? _____
Permitted or not? _____

14. Log off as "User_A" and "User_B". Using your initial user logon name, DROP the two users and the profile you created.

Teradata Training

Notes

Module 47



Priority Scheduler

After completing this module, you will be able to:

- Explain the purpose of Resource Partitions, Performance Groups, and Allocation Groups.
- Determine the minimum % of resources a user can expect with a specific performance group.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Levels of Workload Management.....	47-4
Priority Scheduler Facility	47-6
Priority Scheduler Architecture.....	47-8
Priority Scheduler Architecture with TDWM Workloads	47-10
Priority Scheduler Concepts.....	47-12
Resource Partitions and Performance Groups.....	47-14
Relative Weights	47-16
Performance Periods and Milestones	47-18
CPU Usage Limits with Priority Scheduler	47-20
Use of Performance Groups	47-22
Getting Started with Priority Scheduler	47-24
Schmon Utility	47-26
Schmon Example	47-28
Priority Scheduler Administrator	47-34
Summary	47-36
Review Questions	47-38

Levels of Workload Management

The facing page illustrates three tiers of workload management. This module provides details on Priority Scheduler. The Teradata DWM, Teradata QS, and Database Query Log facility are covered in other modules.

Teradata Dynamic Workload Manager (TDWM)

Teradata Dynamic Workload Manager (also known as Teradata DWM or TDWM) provides a graphical user interface (GUI) for creating rules that manage database access, increase database efficiency, and enhance workload capacity. Via the rules created through Teradata DWM, queries can be rejected, throttled, or executed when they are submitted to the Teradata Database.

Teradata Query Scheduler (QS) is designed to provide a facility to submit Teradata SQL jobs to the Teradata Database. TQS is not shown on the facing page, but is an external tool that simply submits SQL to Teradata.

Priority Scheduler

The Priority Scheduler is a resource management tool that controls how compute resources (e.g., CPU) are allocated to different users in a Teradata Database system. This resource management function is based on scheduler parameters that satisfy your site-specific requirements and system parameters that depict the current activity level of the Teradata Database system. You can provide Priority Scheduler parameters to directly define a strategy for controlling compute resources.

Database Query Log

The Database Query Log (DBQL) is a feature that lets you 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.



Levels of Workload Management

Three Tiers of Workload Management

Teradata Dynamic Workload Manager (TDWM)	Pre-Execution
Priority Scheduler	 Query Executes
Database Query Log	Post-Execution

TDWM

Control what and how much is allowed to begin execution.

Priority Scheduler

Manage the level of resources allocated to different priorities of executing work.

Database Query Log

Analyze query performance and behavior after completion.

Priority Scheduler Facility

The Priority Scheduler Facility (PSF) provides a resource partition hierarchy that allows you to control system resources, specifically the CPU resource. With this utility, processes have an externally assigned priority associated with their database session that the Priority Scheduler Facility uses to allocate CPU and I/O resources. Characteristics include:

- Automatic change of priority if needed
 - Time of day
 - Resource usage at the session or query level
- All work in the database treated equal
 - Not biased toward the short and the quick
 - No punishment for the lengthy
- Flexible: When activity is sparse, lower priority jobs get more resources.
- Offers utilities to define scheduling parameters and to monitor your current system activity.

Why create a customized priority environment?

- Assign very high priority users to a very high priority level to support Active Data Warehouse applications.
- Establish priorities to control the impact of TPump load jobs on short, medium and long DS queries that are running at the same time.
- Create a consistent service level for web requests supported in a database doing a mix of decision-making.
- Provide better service for your more important work.
- Control resource sharing among different applications.
- Automate changes in priority by time of day or by amount of CPU used.
- Place a ceiling on Teradata Database system resources for specific applications.



Priority Scheduler Facility

- Teradata's facility to mandate how database resources will be shared.
 - It is a weight-based system that uses relative weights to control how frequently and quickly CPU resources will be available for different categories of work.
 - Does not provide more capacity, but simply allows more control over how the available capacity is used.
- Runs independently on each node in the configuration.
 - Accumulates CPU usage and I/O demand independently on each node and is not coordinated across nodes.
- Priority can automatically change if needed (defined by performance periods or milestones).
 - Time of day
 - CPU Resource usage at query or session level
- Should be used AFTER application-level tuning has been done.
- To configure Priority Scheduler, use:
 - `schmon` command-line utility
 - Priority Scheduler Administrator (PSA) via Teradata Manager
 - Teradata Dynamic Workload Manager (TDWM) when TDWM workloads are enabled

Priority Scheduler Architecture

Priority Scheduler is a resource management tool that controls the dispersal of computer resources in a Teradata Database system. This resource management tool uses scheduler parameters that satisfy site-specific requirements and system parameters that depict the current activity level of the Teradata Database system. You can provide Priority Scheduler parameters to directly define a strategy for controlling computer resources.

The Priority Scheduler does the following:

- Allows you to define a prioritized weighting system based on user logon characteristics.
- Balances the workload in your data warehouse based on this weighting system.

Priority Scheduler includes default parameters that provide four priority levels with all users assigned to one level using performance groups of L, M, H, and R. Additional performance groups can be created and a performance group (e.g., DM) is specified as part of the Account ID.

If TDWM workloads are NOT enabled, when a SQL request is issued, it enters Teradata, is parsed, and broken into steps. Each step is then dispatched as one or many individual processes. Each process active on behalf of a query executes at the same priority. The priority will depend on how the administrator has put together the several components in the priority framework.

V2R6 Priority Scheduler Changes

A number of enhancements have been made with Priority Scheduler with Teradata V2R6.

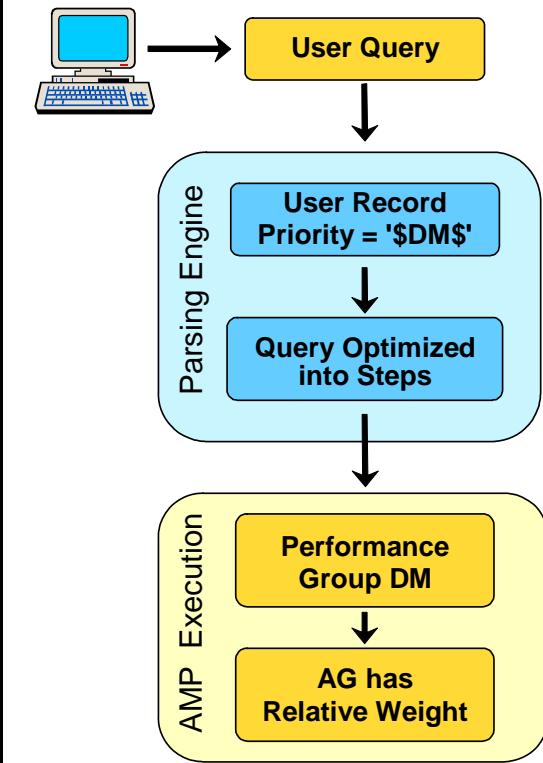
Several options/parameters have been removed, including:

- Internal Performance Groups
- Attributes such as Priority ON/OFF, I/O Prioritization, Throughput vs. Response
- The VALUE parameter of the performance group, which in V2R5 controlled ranking within a resource partition (0-7)
- The allocation group POLICY (default, immediate, relative, absolute)
- The limit on the number of performance groups within one resource partition (was 8 in V2R5)

New enhancements include:

- DBS-generated critical work has been disassociated from the default resource partition. This internal critical work now runs outside the user-controlled priorities as “system” work, and means you no longer have to keep RP0 as the highest-weighted resource partition.
- New allocation group parameter to limit CPU given to an AG.
- Default time quantum for UNIX is now 10 ms., was 20 ms.

Priority Scheduler Architecture



If TDWM workloads are not enabled, priorities are assigned by Priority Scheduler in this manner.

- User Logs on a session with an **Account ID**.
 - Access of USER record confirms **Account ID** which establishes the Performance Group.
- The "priority" for the session is effectively determined by the **Account ID**.
- A query is submitted and the Optimizer breaks query into processing steps.
 - Each step is sent to the AMP as a process, belonging to the User's Performance Group (PG).
 - **Each PG associates the process to one Allocation Group (AG).**
 - The relative weight in the AG controls the priority of the process.

Priority Scheduler Architecture with TDWM Workloads

TDWM (Teradata Dynamic Workload Manager) allows for the creation of workloads. This capability is part of a new concept called **Teradata Active System Management** (TASM). TASM is made up of several products/tools that assist the DBA or application developer in defining and refining the rules that control the allocation of resources to workloads running on a system. These rules include filters, throttles, and “workload definitions”.

The concept of workloads is new with Teradata V2R6.1. To create workloads, you need to use TDWM Release 6.1 (or later) with the following Teradata releases.

- UNIX MP-RAS – Teradata V2R6.0.2 or later
- Windows 2003 or Linux – Teradata V2R6.1.0 or later

TASM and workload definitions will be covered in more detail later in this course.

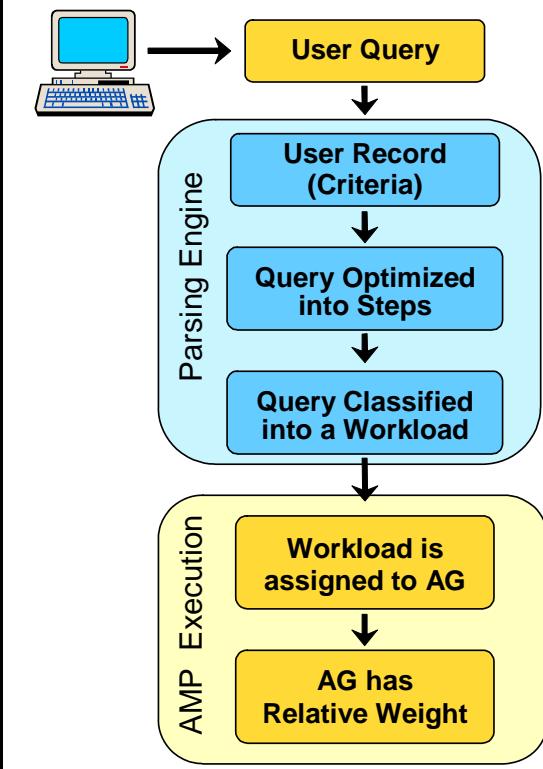
What is a Workload Definition?

A workload represents a portion of the queries that are running on a system. A Workload Definition (WD) is a workload grouping and its operating rules to assist in managing queries. The requests that belong to the same workload will share the same resource priority and exception conditions. It consists of:

- Classification Criteria: criteria to determine which queries belong to the workload. This criteria defines characteristics which are detectable prior to query execution. This is also known as the "*who*", "*where*", and "*what*" criteria of a query. For example, "*who*" may be an account name, "*where*" is the database tables being accessed, and "*what*" may be the type of statement (UPDATE) being executed.
- Exception Criteria: criteria to specify “abnormal” behavior for queries in this workload. This criterion is only detectable after a query has begun execution. If an exception criterion is met, the request is subject to the specified exception action which may be to lower the priority or abort the query.
- Operating Periods: a description of hours of the day and/or days of the week (or month). Directives may be specified for exception handling and Priority Scheduler settings can be changed for each operating period.

A Workload Definition is mapped to an Allocation Group (AG) of Priority Scheduler.

Priority Scheduler Architecture with TDWM Workloads



If TDWM workloads are enabled, priorities are assigned by Priority Scheduler in this manner.

- User Logs on a session.
- A query is submitted and the Optimizer breaks query into processing steps.
- Queries are classified based on criteria (who, where, and what) and the query is assigned to a **Workload**.

Different types queries (e.g., Tactical and DSS) from the same user can automatically be assigned to different workloads and effectively different priorities.

- **Each Workload is assigned to an Allocation Group (AG).**
- Each step is sent to the AMP as a process, effectively associated with an Allocation Group (AG).
- The relative weight of the AG controls the priority of the process.

Priority Scheduler Concepts

To establish the partition hierarchy, establish values for:

- Resource partitions (up to 5)
- Performance groups (up to 40)
- Performance periods (up to 8 per performance group)
- Allocation groups (up to 200)

Resource Partitions

A Resource Partition is a collection of prioritized Performance Groups. A Resource Partition carries a weight that will be compared to other Resource Partition weights.

You can divide your system and user base into resource partitions (RP) that you usually distinguish by use or by accounting strategy. You must number and name the partitions, and assign a weight to each partition that determines the total system resources they receive.

Performance Groups

You define Performance Groups within each Resource Partition. The Performance Group names are used in user Account IDs and determine the priority level for the user.

- Performance Group names match the Account ID string and must be unique.

The default resource partition has a set of 4 performance groups (L, M, H, and R). You can define as many performance groups in a resource partition as you wish. However, the total number of performance groups that can be defined in the system is 40.

Performance Periods

Performance periods link a performance group to an allocation group. Up to 8 performance periods can be assigned to one performance group. Performance periods are based on milestones and are one of three types.

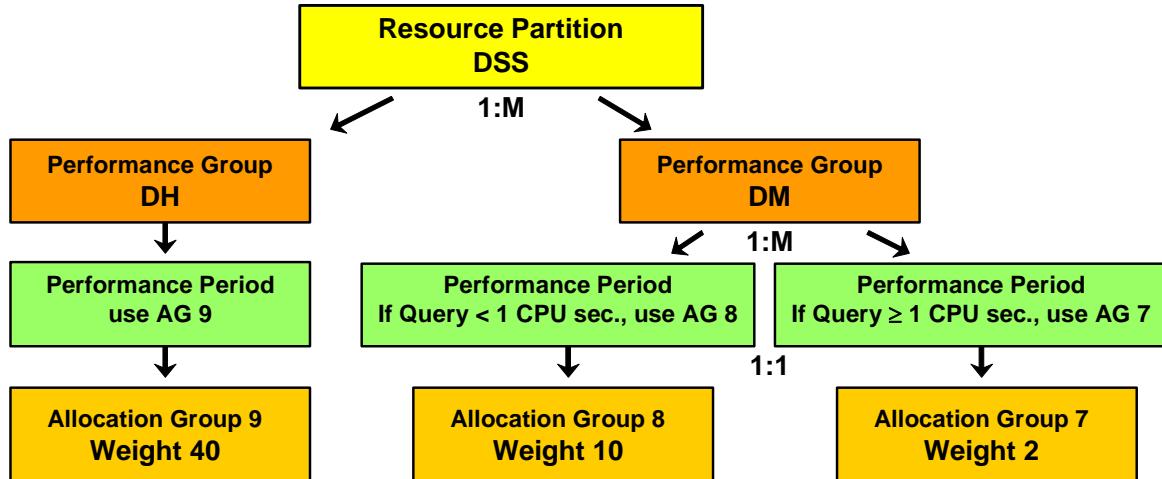
- T for time-of-day
- Q for query resource usage
- S for session resource usage

Allocation Groups

An allocation group (AG) defines a weight that is compared to other Allocation Group weights. These weights determine the amount of scheduling resources allocated. Allocation groups may be referenced by more than one performance period and performance group. However, all references to an allocation group must come from the same resource partition.



Priority Scheduler Concepts



- Maximums of 5 Resource Partitions, 40 Performance Groups, and 200 Allocation Groups
 - Starting with V2R6, all Performance Groups can be placed in a single resource partition (flattened approach) and may be easier to manage.
- Maximum of 8 Performance Periods per Performance Group
 - One Performance Period is assigned to one Allocation Group (1:1 relationship)
- User accounts specify performance group names – for DH, specify '\$DH\$' in account id.

Resource Partitions and Performance Groups

Teradata comes with Resource Partition 0 defined for customer use. Many customers may only need Resource Partition 0 and will not need to define additional Resource Partitions.

Resource Partition 0 uses the standard Teradata priorities – L, M, H, and R. The default allocation weights are shown on the facing page. These priorities are used at the beginning of the Account IDs and are specified as: \$L, \$M, \$H, and \$R.

The complete description of Resource Partition 0 is shown below for Teradata V2R6.0.

Perf Group #	Perf Group Name	Alloc Group #	Alloc Group Weight
0	L	1	5
1	M	2	10
2	H	3	20
3	R	4	40

Note: With Teradata V2R5.1 (and previous release), RP# 0 is also used by internal Teradata software (rollbacks, deadlock detection, etc.). Therefore, RP# 0 is usually given the highest resource partition weight with V2R5.1 and previous releases of Teradata. Starting with Teradata V2R6.0, internal work is no longer assigned to the Default partition.

What is a weight?

Weights (not percentages) are assigned at Resource Partition level and to Allocation Groups within a Resource Partition. Weights are used at the Resource Partition and Allocation Group levels to determine the relative proportion of resources to allocate to the user. Weights are:

- A numeric value used at the Resource Partition Level to compute a relative weight (compared to other Resource Partitions) to determine the proportion of resources the processes of the entire Resource Partition are to receive.
- A numeric value used at the Allocation Group Level to compute a relative weight (within the Resource Partition) to determine the proportion of resources the processes of the Allocation Group are to receive.

Additional Resource Partitions

Additional resource partitions may be added to the Priority Scheduler. The maximum number of resource partitions is 5. The facing page contains an example with a second resource partition named **Tactical**.

For example, you may assign the following Performance Groups to different types of users.

- | | |
|----|--|
| TL | Tactical queries that are all AMP operations – possibly utilize a NUSI |
| TH | Tactical queries that are one or two AMP operations – possibly utilize a PI or USI |



Resource Partitions and Performance Groups

**RP 0 – Default
Weight – 20**

**RP 1 – Tactical
Weight – 60**

**RP 2 – DSS
Weight – 20**

PG #	Name	AG #	Weight	PG #	Name	AG #	Weight	PG#	Name	AG #	Weight
0	L	1	5	4	TL	5	10	6	DL	7	2
1	M	2	10	5	TH	6	40	7	DM	8	10
2	H	3	20					8	DH	9	40
3	R	4	40								

Resource Partition 0 is provided with Teradata.

To create this partition,
schmon -b 1 Tactical 60

To create this partition,
schmon -b 2 DSS 20

Example:

- L, M, H, or R – assigned to Batch and Load jobs
- TL – assigned to Tactical queries that are all AMP operations (e.g., NUSI)
- TH – assigned to Tactical queries that are one or two AMP operations (e.g., PI)
- DM – assigned to unknown DSS queries (e.g., ad hoc)
- DH – assigned to known DSS queries

Assuming queries are active in all 3 partitions, system resources are effectively divided between the three partitions as follows:

Default (20/100) = 20% Tactical (60/100) = 60% DSS (20/100) = 20%

Relative Weights

Resource partition weight is a relative weight, since its value is relative to the weights of the currently active partitions. Dividing the weight of a partition by the sum of the weights of all active partitions gives the percentage of total Teradata Database system resources for that partition. The concept applies to performance groups within a resource partition.

The facing page contains two examples of how the system resource is divided between resource partitions and between performance groups within a resource partition.

Scheduling Policies (Teradata Release V2R5.1 and before)

With Teradata Release V2R5.1 (and before), an allocation group had an option called policy. The choices for policy included **Default**, **Immediate**, **Absolute**, and **Relative**. This option no longer exists starting with TD V2R6.0. However, starting with V2R6.0, an optional maximum percentage can be added following the allocation group weight.

Use the **Default** or **Immediate** scheduling policy unless there is a compelling business reason to use one of the other scheduling policies.

Do you want a ceiling on resource usage? If NO, then use ...

Default (most useful for typical decision support queries)

- Consider both Group and Individual Process usage
- Slow consumers can catch up even if group allocation is temporarily exceeded
- Can use more than the allocation if the system is idle

Immediate (most useful for short work and transactions)

- Consider only the Group usage
- Slow consumers may continue to lag behind
- Can use more than the allocation if system is idle

Do you want a ceiling on resource usage? If YES, then use ...

Absolute (creates an absolute ceiling on usage)

- Consider Group weight as an absolute %
- Absolute % is based on the assigned weighting
- Can never get more, even if machine is idle

Relative (creates a relative ceiling on usage)

- Consider Group weight as relative %
- Relative % is computed based on active groups
- Can never get more, even if machine is idle



Relative Weights

Relative weights are based on the assigned weights of the ACTIVE Allocation Groups.
Fewer ACTIVE AG's will result in more CPU priority for those AG's that are ACTIVE.

	PG #	PG Name	AG #	AG Weight	Active Sessions	Relative Weight Calculation	AG %
RP 0 Weight 20	0	L	1	5	Y	$20/100 \times 5/35$	2.9
	1	M	2	10	Y	$20/100 \times 10/35$	5.7
	2	H	3	20	Y	$20/100 \times 20/35$	11.4
	3	R	4	40	N		
RP 1 Weight 60	4	TL	5	10	Y	$60/100 \times 10/50$	12.0
	5	TH	6	40	Y	$60/100 \times 40/50$	48.0
RP 2 Weight 20	6	DL	7	2	N		
	7	DM	8	10	Y	$20/100 \times 10/10$	20.0
	9	DH	9	40	N		

	PG #	PG Name	AG #	AG Weight	Active Sessions	Relative Weight Calculation	AG %
RP 0 Weight 20	0	L	1	5	Y	$20/80 \times 5/15$	8.3
	1	M	2	10	Y	$20/80 \times 10/15$	16.7
	2	H	3	20	N		
	3	R	4	40	N		
RP 1 Weight 60	4	TL	5	10	N		
	5	TH	6	40	Y	$60/80 \times 40/40$	75.0
RP 2 Weight 20	6	DL	7	2	N		
	7	DM	8	10	N		
	9	DH	9	40	N		

Performance Periods and Milestones

A performance period type defines a type of threshold used to change performance periods for a Performance Group. The milestone limit represents that threshold. The milestone limit triggers an automatic change in Allocation Group when a threshold you define is reached.

You can express milestone limits in the following units:

- Time-of-day – minutes of military time (0 – 2359) and represent time periods during a 24-hour day.
- Session resource usage – defined in CPU seconds (0 – 86400) and represents an amount of session CPU resource consumption per node.
- Query resource usage – defined in CPU seconds (0 – 86400) and represents an amount of Query CPU resource consumption per node.

The facing page contains an example of having multiple performance periods and automated priority changes.

Day-of-Week options for Time-of-Day Milestone

Optionally, you can use a day-of-week specification with a time-of-day milestone for a performance period to indicate days when the performance period is to be active. This specification might indicate one or more individual days, or a range of days, but not both.

Days are numbered sequentially from 0 to 6, Sunday through Saturday. A range of days is indicated by two day numbers separated by a hyphen.

If a day-of-week specification is present for one performance period, then day-of-week must be present for all.

A time-of-day milestone for a performance period defines the end of a time period during which the associated Allocation Group is to control processes. The beginning of each period is the end of the preceding period. This might be on a previous day if day of week has been specified.

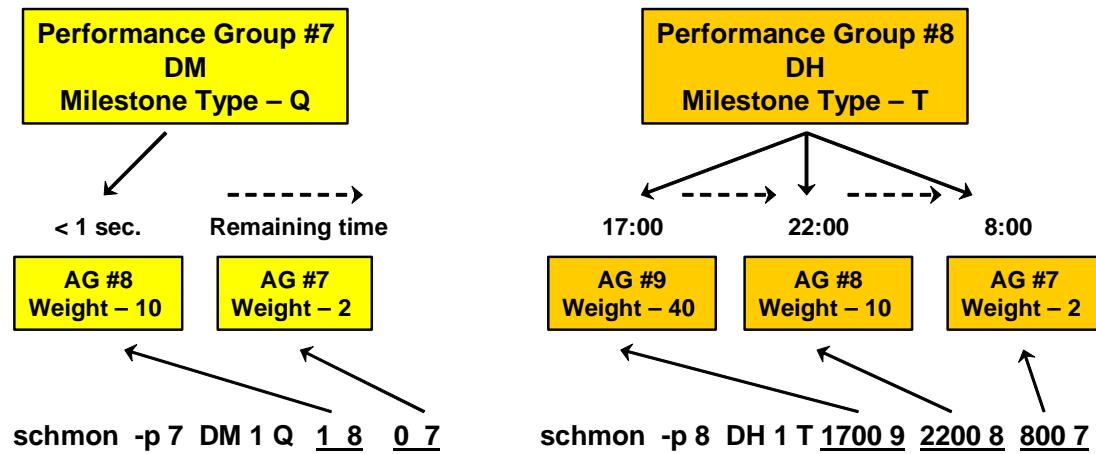
Resource Usage

When you express milestone limits in units of session or query resource usage, their values indicate the total resource usage of all processes working on behalf of a session, or a query submitted by a session, on that node. In this case, a performance period controls processes grouped into sessions. Since some sessions or queries might consume more or less resources than others, several performance periods of a Performance Group might be actively controlling processes of different sessions concurrently.

Performance Periods and Milestones

A performance period type defines a type of threshold (milestone limit) used to change performance periods for a Performance Group. Milestone options are:

- T – Time-of-day
- S – Session resource usage (based on CPU time accumulated in a session by node)
- Q – Query resource usage (based on CPU time accumulated by a query by node)



Milestones cause a change in Allocation Group, not Performance Group.

CPU Usage Limits with Priority Scheduler

It is possible to limit the amount of CPU usage that is available to the system, to a specific resource partition, or to a specific allocation group. A percentage can be included with each of these levels. The general format of the “Limit” follows:

Limit – range is 1 through 100 or the value “none”

- A value of 100 indicates that no limit is to be enforced.
- A character string of “none” indicates that no limit is to be enforced.
- For Resource Partitions and Allocation Groups, if *Limit* is not present, any previously defined limit is removed.

System Level (Node level)

To limit the CPU usage at the system level, you can specify a percentage value to limit the amount of CPU usage available to all Teradata Database sessions. This usage does not include non-Teradata work, such as time-share users, I/O or other interrupt services, Gateway processing, or streams work.

For example, to set the Teradata Database system CPU usage limit to 80%, use the following command:

schmon -I 80 (Sets system CPU % limit to 80%)

Resource Partition Level

To limit CPU usage at the resource partition level, you can specify a percentage limit on total CPU usage by all processes controlled that by a resource partition.

For example, to set a 75% limit at a Resource Partition level, use the following command:

schmon -b 1 Tactical 100 75 (Sets RP CPU % limit to 75%)

Note: The Resource Partition weight is 100 and the CPU limit is 75%.

Allocation Group Level

To limit CPU usage at an allocation group level, you may specify a percentage limit on total CPU usage by all processes controlled by the allocation group.



CPU Usage Limits with Priority Scheduler

CPU usage limits may be used to place a “**ceiling**” on the amount of resources that are available to a specific level.

- **System level**

Example: To limit system CPU usage to 80%,

schmon -l 80 (l: lower case L)

- **Resource Partition level**

Example: To limit CPU usage to 75% for Resource Partition "Tactical",

schmon -b 1 Tactical 60 75
 ↑ ↑
 weight limit

- **Allocation Group level**

Example: To limit CPU usage to 50% for Allocation Group #9,

schmon -a 9 N 40 50
 ↑ ↑
 weight limit

Note: All weights and CPU limits are enforced by Priority Scheduler at the node level.

Use of Performance Groups

Performance Group names are specified within the Account parameter when creating or modifying a user. Performance Group names must be delimited with a “starting \$” and an “ending \$”. For example, DH would be identified as \$DH\$.

In earlier Teradata Database versions, a single character (L, M, H, or R) prefixed by the \$ character in the Account ID string indicated the Performance Group. To provide backward compatibility, Priority Scheduler provides each of these single character identifiers as a Performance Group name within default Resource Partition 0.

In this special case, the four Performance Group names (L, M, H, and R) do not require an ending \$ character in the Account ID string. In this case, the strings \$M and \$M\$ are equivalent.

The logon process assigns each user session to a Performance Group based on the accounted string of the logon command. If a Performance Group cannot be assigned based on the Account ID string, a default assignment is made.

Each session has a designated Performance Group. When a session begins a process, it falls under the control of a performance period whose milestone limit conditions are met.



Use of Performance Groups

Performance Group names are specified within the Account parameter when creating or modifying a user.

```
CREATE USER rjones AS PERM=0, SPOOL=500e6, PASSWORD=rj182130,  
ACCOUNT=('$DM$', '$DH$');
```

If "rjones" only needs 'DM' as a priority, then only include that account ID.

Users log on to Teradata using valid account IDs which can include Performance Group names.

.logon educ1/rjones, rj182130	Uses DM – default performance group
.logon educ1/rjones, rj182130, '\$DHS'	Uses DH – specified performance group

Getting Started with Priority Scheduler

To establish the partition hierarchy, establish values for:

- Resource partitions (up to 5)
- Performance groups (up to 40)
- Performance periods (up to 8 per performance group)
- Allocation groups (up to 200)

Resource Partitions

Divide the computer system and user base into resource partitions (RP) that you usually distinguish by use or by accounting strategy. You must number and name the partitions. Assign a weight to each partition that determines the total system resources they receive.

Allocation Groups

An allocation group (AG) defines a weight and a division type that determines the amount of scheduling resources allocated. Allocation groups may be referenced by more than one performance period and performance group in the same resource partition.

When creating an allocation group, it is necessary to set the division type.

- N for NONE – resources are divided amongst processes; better for complex queries and all-AMP queries. N is the default and recommended choice for most environments.
- S for SESSION – resources are divided by # of active sessions, then by processes; may be better for simpler queries and single-AMP queries. (This option is usually not needed or used as the impact it has is negligible.)

Performance Groups

Each resource partition has a set of performance groups. Performance groups divide the resource partition in priority groups. 4 performance groups per resource partition are typical. However, with V2R6, you can have more than 4 performance groups per resource partition and having all of the performance groups in one partition (flattened approach) may be useful in some environments.

Performance Periods

Performance periods link a performance group to an allocation group. Up to 8 performance periods can be assigned to one performance group.

- Performance periods make it possible to have changes in priority weight by time or resource usage.



Getting Started with Priority Scheduler

- Create and name a Resource Partition and assign a weight.**

Optionally assign all Tactical queries to a RP with the highest priority.

- Create Allocation Groups.**

Specify weight and division type (N or S).

N – None (default & best choice for most environments)
S – Session (normally not used)

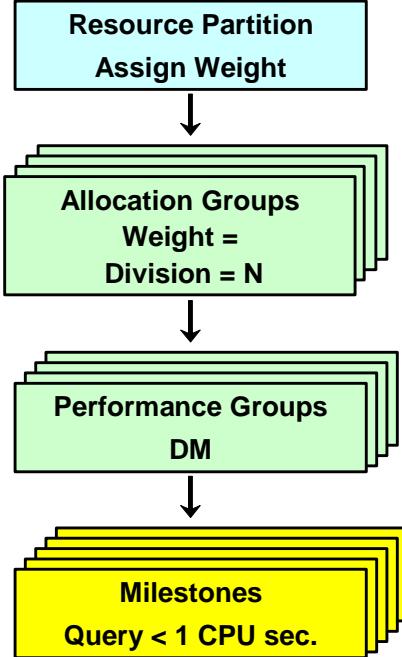
- Create Performance Groups with unique names.**

Assign an Allocation Group to the Performance Group or to each Performance Period.

Optionally define Performance Periods or Milestones. If weight will not change, Session = 0.

- Modify users to associate Account IDs with unique Performance Group names – ex. '\$DM\$'.**

- Use Performance Group names on logons.**



Schmon Utility

The **schmon** utility is used to add and change Priority Scheduler Facility parameters. There are both command-line and an X-Windows versions of this utility.

Schmon – a command-line interface.

Xschmon – a graphical user interface that uses the X-Window system.

The section on the Priority Scheduler Facility in the *Teradata Database User Utilities* reference manual has details for all parameters to all commands you can execute with this utility.

Via Teradata Manager, you can also use the Priority Scheduler Administrator (PSA) utility to manage priority scheduler settings. This is a Windows based utility and provides an easier to use GUI interface.



schmon Utility

- The schmon utility modifies and displays scheduler parameters and displays current scheduling activity.
- schmon -h will display help information about the options.
- schmon -d will display ...
 - current settings including the age and active times
 - Allocation Group to Performance Group assignments
 - Allocation Group weight and policy settings
- schmon -m or -M will display the current scheduling activity.
 - This shows the current CPU and I/O usage, the number of processes attached to each Allocation Group and other data.
- Priority Scheduler Administrator (PSA) provides a Windows interface to also manage the Priority Scheduler facility.

Schmon Example

The facing page contains the **schmon** parameters needed to establish the Tactical resource partition shown previously.

Some additional notes:

When creating an allocation group, it is necessary to set the division type.

- N for NONE - Resources are divided amongst processes; better for complex queries and all-AMP queries. N is the default and recommended value for most environments.
- S for SESSION – Resources are divided by # of active sessions, then by processes; better for simpler queries and single-AMP queries.

The example on the facing page shows the division type set to N.

When creating a Performance Group, either T (time), S (session), or Q (Query) is used as the Performance Period. The last milestone for the Session or Query group type must be 0.

Following the T, S, or Q, you can include up to 8 performance period pairs consisting of milestone limits and allocation group #.

To modify the weight of an existing component, use the same commands except with a different weight. The change will be effective immediately. For example:

```
# schmon -a 7 S 25 none (changes AG# 7 weight to 25 and removes the limit)
```

```
# schmon -b 1 Tactical 80 none (sets RP# 1 weight to 80 and removes the limit)
```



schmon Example

The following commands will establish the Tactical resource partition shown earlier.

To create Resource Partitions:

# schmon -b 0	Default	20	100	
# schmon -b 1	Tactical	60	100	
# schmon -b 2	DSS	20	100	
	RP#	RP Name	Weight	(Optional % Limit)

To create Allocation Groups:

# schmon -a 5	N	10	100	
# schmon -a 6	N	40	100	
# schmon -a 7	N	2	100	
# schmon -a 8	N	10	100	
# schmon -a 9	N	40	100	
	AG#	Division Type (N or S)	Weight	(Optional % Limit)

To create Performance Groups:

# schmon -p 4	TL	1	S	0	5								
# schmon -p 5	TH	1	S	0	6								
# schmon -p 6	DL	2	S	0	7								
# schmon -p 7	DM	2	Q	1	8	0							
# schmon -p 8	DH	2	T	1700	9	2200	8	800	7				
	PG# (unique)	PG Name (unique)	RP#	Session, Query, Time	Mile-stone	AG#	Mile-stone	AG#	Mile-stone	AG#			

Schmon -d Example Output

An example of **schmon -d** option is shown on the facing page.

Options for **schmon** include:

schmon

-a ['all' <AG#>] [<AG#> -x] [<AG#> -s -S] [<AG#> <div> [X] <weight> [limit]]	allows you to set/display allocation groups.
-b ['all' <RP#>] [<RP#> -x] [<RP#> -s -S] [<RP#> <RPNAME> <weight> [limit]]	allows you to set/display resource partitions.
-p ['all' <PG#>] [<PG#> -x] [<PG#> -s -S] [<PG#> <PGNAME> <R> <T> <PP(i)(i = 0,7)>]	allows you to set/display performance groups.
-c [-p -b -a -w -t] [-f path]	allows you to dump current settings as commands.
-d	displays the current settings in a multiple line format.
-f [path]	specifies a path for which input is to be read.
-h [specific option(s)]	displays help information for the option specified.
-l [limit]	sets the system CPU usage limit.
-m [-S [delay [reps]]]	monitors PS statistics for the current node.
-M [-p] [delay [reps]]]	monitors PS statistics for the current node.
-s ['all' <id>] [-S] [-S]	displays PS data for specified sessions.
-t <age> <active> <disp> <iocconc>	displays Age, Time, Active Time, and Disp. Time.
-w <reserve> <maximum>	sets/displays the number of processes available for AG with the Expedite attribute.
-X [-p -b -a -w -t]	dumps all the fields from GDO in hexadecimal format.



schmon -d Example Output

Scheduler Times & Attributes:

Age Time(sec): 60.0 Active Time(sec): 61.0 Limit(%): none I/O Concurrency: 10

Resource Partitions (0 - 4)

Id	Partition Name	Weight	Limit	
0	Default	20	none	
1	Tactical	60	none	(Note: Limit of 100% is identified as "none".)
2	DSS	20	none	

Performance Groups (0 - 39)

Id	Group Name	RP	Type	Milestones & Alloc Groups[0-4]
0	L	0	S	0.00 1
1	M	0	S	0.00 2
2	H	0	S	0.00 3
3	R	0	S	0.00 4
4	TL	1	S	0.00 5
5	TH	1	S	0.00 6
6	DL	2	S	0.00 7
6	DM	2	Q	1.00 8 0.00 7
7	DH	2	T	0800 7 1700 9 2200 8

Allocation Groups (0 - 199)

Id	Type	Weight	Limit	(new starting with V2R6)
1	N	5	none	
2	N	10	none	
3	N	20	none	
4	N	40	none	
5	N	10	none	
6	N	40	none	
7	N	2	none	
8	N	10	none	
9	N	40	none	

AWT Expedited work type limits (new starting with V2R5)

res	max
0	80

Schmon -m Example

Examples of the **schmon -m** are shown on the facing page.

An example of **schmon -m** with a single resource partition is shown below. The **-M** option (not shown) provides information for all of the SMPs or nodes.

Answer to question on facing page:

Queries running between 17:00 (5:00 PM) and 22:00 (10 PM) in the DH group use AG #8.

Column definitions for AG #3 at the bottom of the facing page are:

- **AG:** The ID of each active allocation group. Only allocation groups that have seen new requests for AMP worker tasks during the most recent age interval (usually the last 60 seconds) will appear in this list.
- **#requests:** This column reflects the number of messages that have been received for this allocation group within the age interval. This represents work that needs to acquire AMP worker tasks to get underway. This may be work belonging to any work type.
- **Avg queue wait (msec):** This column shows the average time per re-request spent waiting for an AMP worker task, in milliseconds, as captured during the age interval. On UNIX, if this is a number from 0 to 5, it is considered normal and is not pointing to a performance issue.
- **Avg queue length:** This represents the average queue length during the age interval for new requests that were waiting in line to be serviced. A zero in this column means that on average, there was no line of requests waiting for an AWT. Since this is an average, and only whole numbers are represented, a zero could represent a fraction.
- **Avg service time (msec):** This is the average amount of time an AMP worker task was held within the recent age interval. This represents wall-clock time, not milliseconds of CPU consumption, as is reported for CPU usage in the higher portion of the schmon -m output.



schmon -m Example

System with 2 Resource Partitions being utilized:

```

schmon -m
Stats: Wed Mar 25 19:15:14 2008

      Rel    Avg CPU      Avg I/O      # of      # of
      RP Wgt   % (msec)   % (sblk) Procs Sets
===== ===== ===== ===== ===== ===== ===== =====
      0  50    0     311     7     597      8      2
      2  50    2    4718    91   13135      4      2

      Rel    Avg CPU      Avg I/O      # of      # of
      AG Wgt   % (msec)   % (sblk) Procs Sets Performance Groups Affected
===== ===== ===== ===== ===== ===== ===== ===== =====
      2   8    0     175     5     393      7      1 M
      3  16    0     136     2     204      1      1 H
      8  24    5    3430    26    4717      2      1 DM, DH
      9  48    2    1288    65    8418      2      1 DM, DH

System: %CPU  CPU(msec) I/O(sblk) #procs
===== ===== ===== ===== =====
      1      5029     13722      37

      Avg queue  Avg queue  Avg service
      AG #requests wait(msec) length    time(msec)
===== ===== ===== ===== ===== =====
      3           1        57         0       6832
  
```

Question:
At this time, what AG is used for queries running with DH priority?

Priority Scheduler Administrator

The Teradata Priority Scheduler Administrator (PSA) is a resource-management tool that provides an easy-to-use graphical interface that allows you to define Priority Definition Sets, generate schmon scripts to implement these sets, and monitor and control the Priority Scheduler environment.

This utility is accessed via Teradata Manager. An example of a PSA display representing the Tactical resource partition is shown on the facing page.

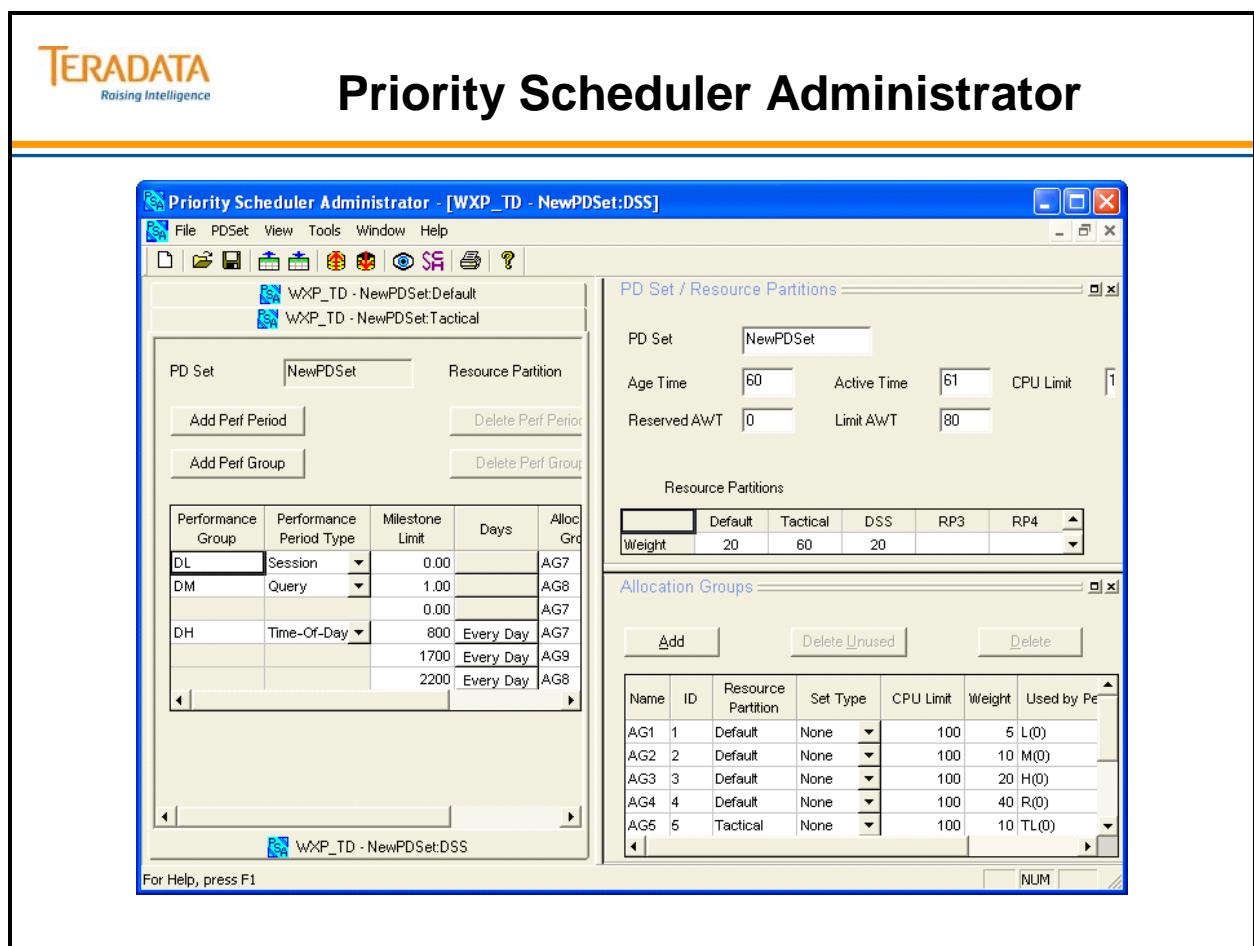
Features of PSA include:

- Makes Priority Scheduler more usable and understandable.
- Provides visualization of Resource Partition and Allocation Group weights and CPU Utilization.
- Eliminates much of your current guesswork about the results of Priority Scheduler changes.
- Easily manage multiple Priority Scheduler configuration profiles.

Corresponding schmon Commands

The following “schmon” commands correspond to the PSA example shown on the facing page.

```
schmon -b 0 Default 20 100
schmon -b 1 Tactical 60 100
schmon -b 2 DSS 20 100
schmon -a 5 N 10 100
schmon -a 6 N 40 100
schmon -a 7 N 2 100
schmon -a 8 N 10 100
schmon -a 9 N 40 100
schmon -p 4 TL 1 S 0 5
schmon -p 5 TH 1 S 0 6
schmon -p 6 DL 2 S 0 7
schmon -p 7 DM 2 Q 1 8 0 7
schmon -p 8 DH 2 T 1700 9 2200 8 800 7
```



Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Priority Scheduler is Teradata's facility to mandate how database resources will be shared.
 - It is a weight-based system that uses relative weights to control how frequently and quickly CPU resources will be available for different categories of work.
- Priority can automatically change if needed (defined by performance periods or milestones).
 - Time of day
 - CPU Resource usage at query or session level
- To configure Priority Scheduler, use:
 - schmon command-line utility
 - Priority Scheduler Administrator (PSA) via Teradata Manager
 - Teradata Dynamic Workload Manager (TDWM) when TDWM workloads are enabled

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. Given the following, what is minimum % of resources that the following Performance Groups (PG) can expect.

TAC_M = _____ DSS_H = _____

Default		Tactical		Standard	
<u>RP 0 - Weight 20</u>		<u>RP 1 - Weight 60</u>		<u>RP 2 - Weight 20</u>	
PG Name	Wgt	PG Name	Wgt	PG Name	Wgt
L	5	TAC_L	5	DSS_L	5
M	10	TAC_M	10	DSS_M	10
H	20	TAC_H	30	DSS_H	30
R	40	TAC_R	55	DSS_R	75

2. Without TASM workloads enabled, a user session is associated with a _____ which is effectively assigned to an _____.
3. With TASM workloads enabled, a user query is associated with a _____ which is effectively assigned to an _____.

Teradata Training

Notes

Module 48



System Access Controls

After completing this module, you will be able to:

- **Describe where and how to control and log user access to the Teradata database.**
- **Use views and macros to limit user access to data.**
- **Design your system hierarchy structures for better security and easier maintenance.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

System Access Control Levels.....	48-4
Teradata Password Encryption.....	48-6
Password Security Features	48-8
Teradata Connectivity	48-10
Sessions and Session Pools	48-12
Teradata Director Program (TDP)	48-14
TDP Exits	48-16
Host Logon Processing	48-18
Objects used in Host Logon Processing.....	48-20
GRANT/REVOKE LOGON Statements	48-22
GRANT/REVOKE LOGON Example	48-24
Session Related Views	48-26
DBC.LogonRules View	48-28
DBC.LogOnOff View	48-30
DBC.SessionInfo View	48-32
Additional Utilities to View Sessions	48-34
Teradata Manager Sessions.....	48-36
Query Session Utility	48-38
Access Control Mechanisms	48-40
Using Views to Limit Access.....	48-42
Using Macros to Reduce User SQL Complexity	48-44
Structure the System	48-46
A Recommended Access Rights Structure	48-48
A Recommended Structure Using Roles.....	48-50
A Recommended System Hierarchy	48-52
System Access Controls Summary	48-54
Review Questions	48-56

System Access Control Levels

The mission of security administration on a Teradata system is to:

- Prevent unauthorized persons from gaining access to the RDBMS and its resources.
- Permit legitimate users access to only those resources they are authorized to use.

A variety of mechanisms provide security to the data stored on a Teradata system.

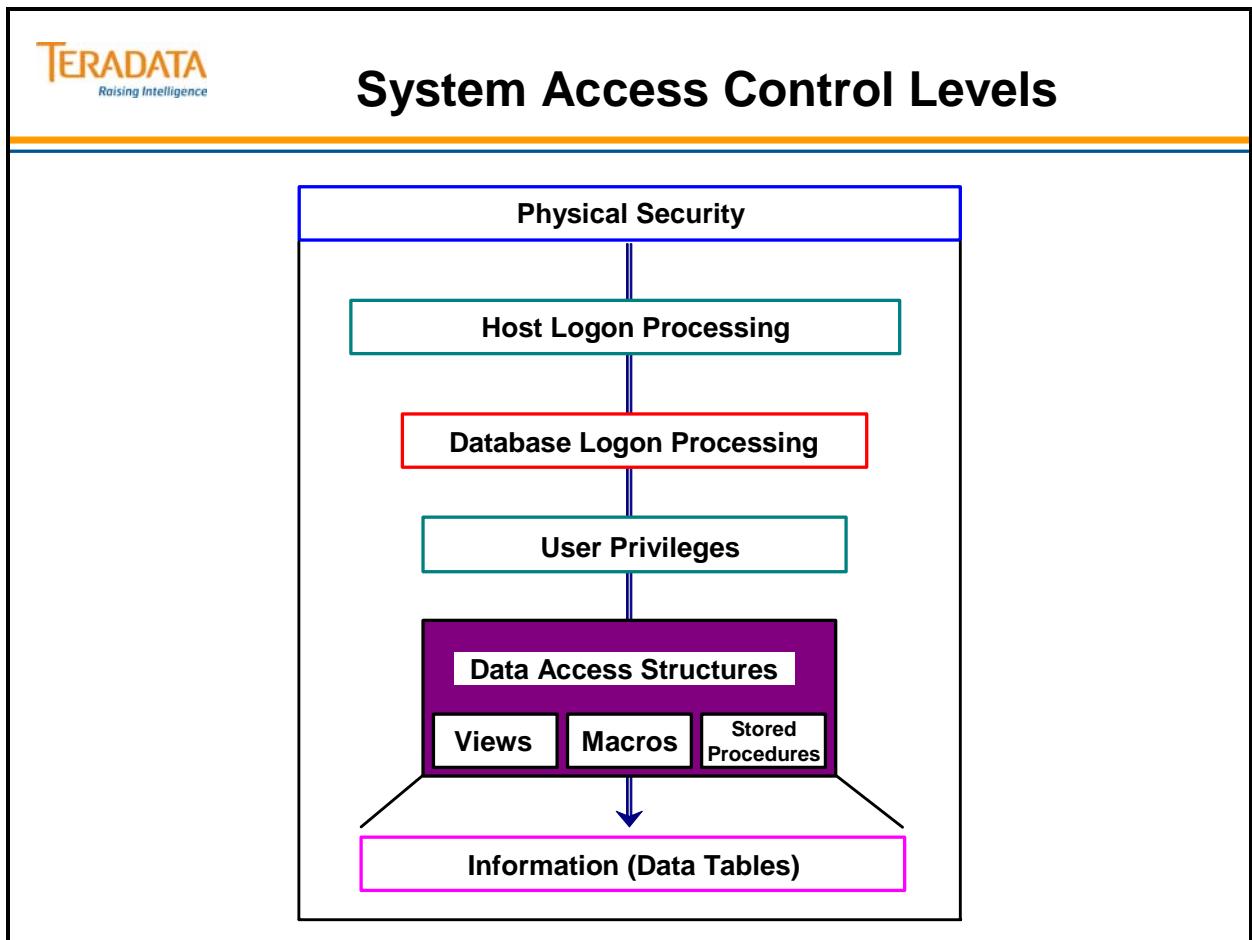
Access Control Levels

This lesson introduces a guideline for how to determine user access rights and explains how Teradata verifies user access rights.

There are three levels of access controls for the Teradata database:

Physical Security	Physical security pertains to the actual building or computer room in which the Teradata system resides. The system's owner designs and implements physical security.
Host Logon Processing	Host logon processing is the first level of access control and allows or disallows connection between the host and database systems. It involves a host ID and password. This level controls access to the host system.
Database Logon Processing	Database logon processing is the second level of access control and determines access to the Teradata system. This level employs a username and password. It controls access to the Teradata system itself.

Data access structures (views, macros and tables) are discussed later in this module.



Teradata Password Encryption

You can give access to the Teradata database with the CREATE USER statement, which identifies a username and, usually, a password value.

Although the username is the basis for identification to the system, it is not usually protected information. Often the username is openly displayed during interactive logon, on printer listings, and when session information is queried.

To protect system access, associate a password with the username. Teradata does not display or print passwords on listings, terminals or PC screens.

Note: Neither you nor other system users should ever write down passwords or share them among users.

Teradata stores password information in encrypted form in the DBC.Dbase system table. Information stored in the table includes the date and time a user defined a password, along with the encrypted password. As the administrator, you may modify passwords temporarily when the PasswordLastModDate plus a fixed number has been reached. This allows you to ensure that users change their passwords regularly.

To establish a session on the Teradata system, a user must enter a username at logon. Upon successful logon, the username is associated with a unique session number until the user logs off.

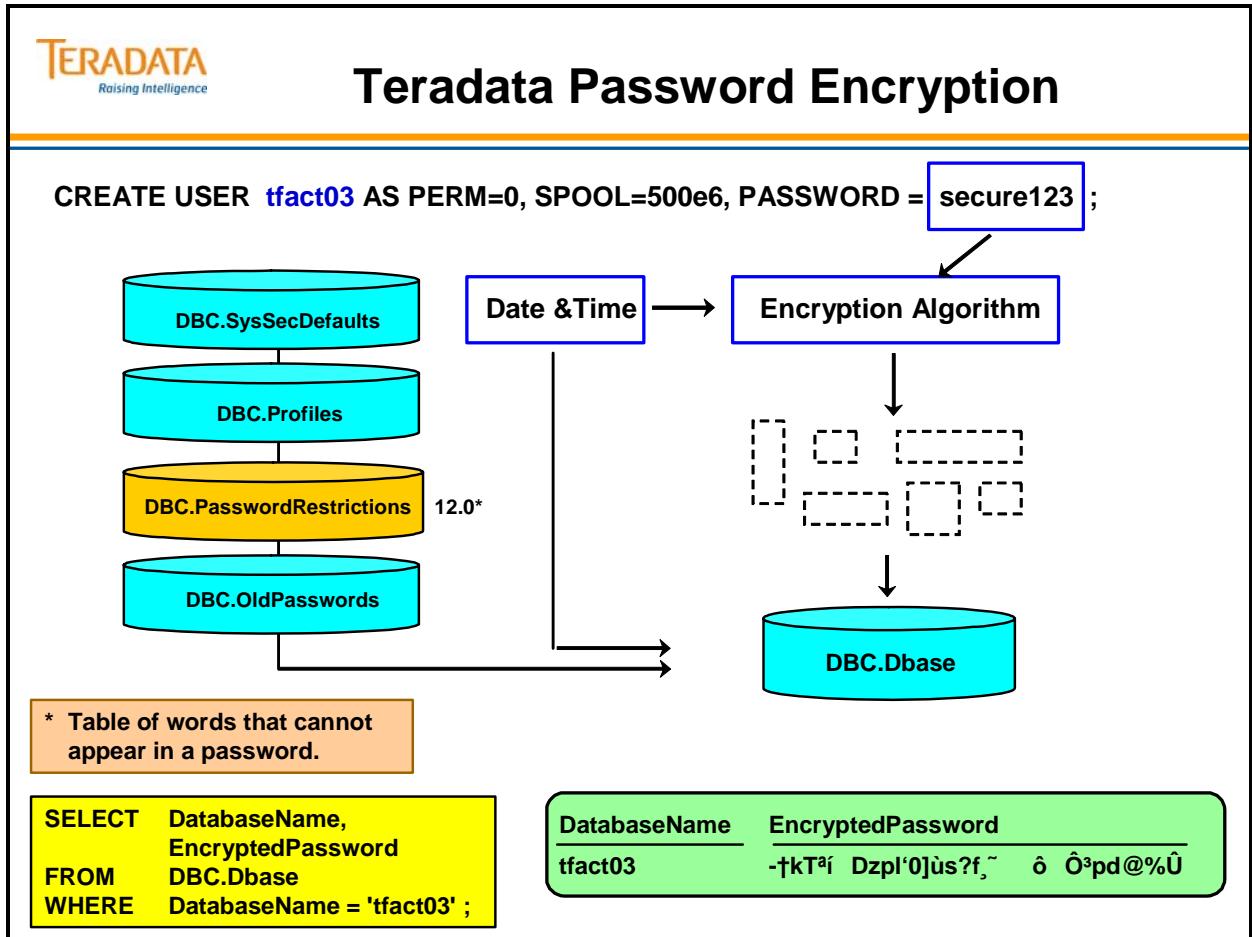
To supervise and enforce users' access rights to stored data, the system associates each username with a default storage area and an arrangement of access rights.

Displaying Passwords

The PasswordString column from the DBC.Dbase table displays encrypted passwords. The SQL request on the facing page demonstrates how you can access an encrypted password. Note that a password cannot be decrypted.

DBC.Users View

The DBC.Dbase table stores the date and time a user defines or modifies a password. The DBC.Users[V] view displays PasswordLastModDate and PasswordLastModTime. A user can modify his or her password without additional access privileges.



Password Security Features

Teradata password security features allow you to:

- Expire passwords after a specific number of days.
- Define the amount of time to elapse before a password can be reused.
- Control minimum/maximum length of password.
- Disallow digits/special characters in a password.
- Limit the number of erroneous logon attempts before the system locks a user's access.
- Automatically unlock users after a specific period of time.

You can enable these features by updating the appropriate row in the DBC.SysSecDefaults table as shown on the facing page. The DBC.SecurityDefaults[V] view can also be used to view/update this table. After modifying this table, it is necessary to restart Teradata for the changes to be in effect.

When you create a new user, you also create a temporary password for the user. When the user logs on for the first time, he or she is prompted to change the password.

If a user forgets the password, you can assign a new temporary password. [As another option, you can set user passwords not to expire.]

If you attempt to set the PasswordMinChar attribute equal to 0, Teradata will assume a value of 1.

Note: If MaxLogonAttempts is set to a value other than zero, and if the time interval for locking users after erroneous attempts is left at zero, then the user is never locked.

Starting with Teradata V2R6.1, options that can be placed in the PasswordSpecChar column include:

Option PasswordSpecChar	N	Y	A	B	C	D	E	F	G	H	I	J	K	L	M	O	P	R
RULE Username	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
RULE Upper/ Lower	Y	Y	Y	Y	Y	Y	R	R	Y	Y	Y	Y	Y	Y	R	R	R	
RULE One Alpha	Y	Y	Y	R	R	R	R	R	Y	Y	Y	R	R	R	R	R	R	
RULE Spec Chars	N	Y	R	N	Y	R	N	Y	R	N	Y	R	N	Y	R	N	Y	R



Password Security Features

The **DBC.SecurityDefaults[V]** (view) can be used to view/update **DBC.SysSecDefaults** table.

ExpirePassword	Number of days to elapse before the password expires. Zero (0) indicates passwords do not expire; default is 0.
PasswordMinChar	Minimum number of characters in a valid password string; default is 1.
PasswordMaxChar	Maximum number of characters in a valid password string; default is 30.
PasswordDigits	Indicate if digits are to be allowed in the password (Y, N, or R); default is Y; (R – one or more digits are required in password).
PasswordSpecChar	Indicate if special characters are allowed in the password (Y or N); default is Y; (Options – A to P, R – options provide for more secure passwords).
PasswordRestrictWords	Indicate whether or not a password is subject to the content restrictions (Y or N); default is N.
MaxLogonAttempts	Number of erroneous logons allowed before locking user. Zero (0) indicates that user is never locked; default is 0 - max is 32,767.
LockedUserExpire	Number of minutes to elapse before a locked user is unlocked. Zero (0) indicates immediate unlock; -1 = locked indefinitely; default is 0 - max is 32,767.
PasswordReuse	Number of days to elapse before a password can be reused. Zero (0) indicates immediate reuse; default is 0 - max is 32,767.

To change system-wide password security features:

1. UPDATE this view or table with the desired values
2. Restart Teradata (required)

Teradata Connectivity

Teradata utilities and software programs support Teradata database access in both mainframe and LAN environments. Utilities and programs run under the client's operating system and provide the functionality for a user to access the database system.

When a system is configured, host numbers are assigned to different channel and LAN connections. It is possible to enable/disable user access from specific host numbers.

Channel Environment

The Teradata Channel Interface is an architecture that enables communication between a mainframe client and a Teradata server using a channel with either a parallel (Bus/Tag) or serial I/O (ESCON) interface.

With Teradata servers, the SMP nodes use I/O adapters such as the PBSA (PCI Bus ESCON Adapter) to connect to an ESCON channel or the PBCA (PCI Bus Channel Adapter) to connect to a Bus/Tag channel. The TDP software executing on the mainframe communicates with the PE software executing on the SMP.

LAN Environment and Teradata Gateway Software

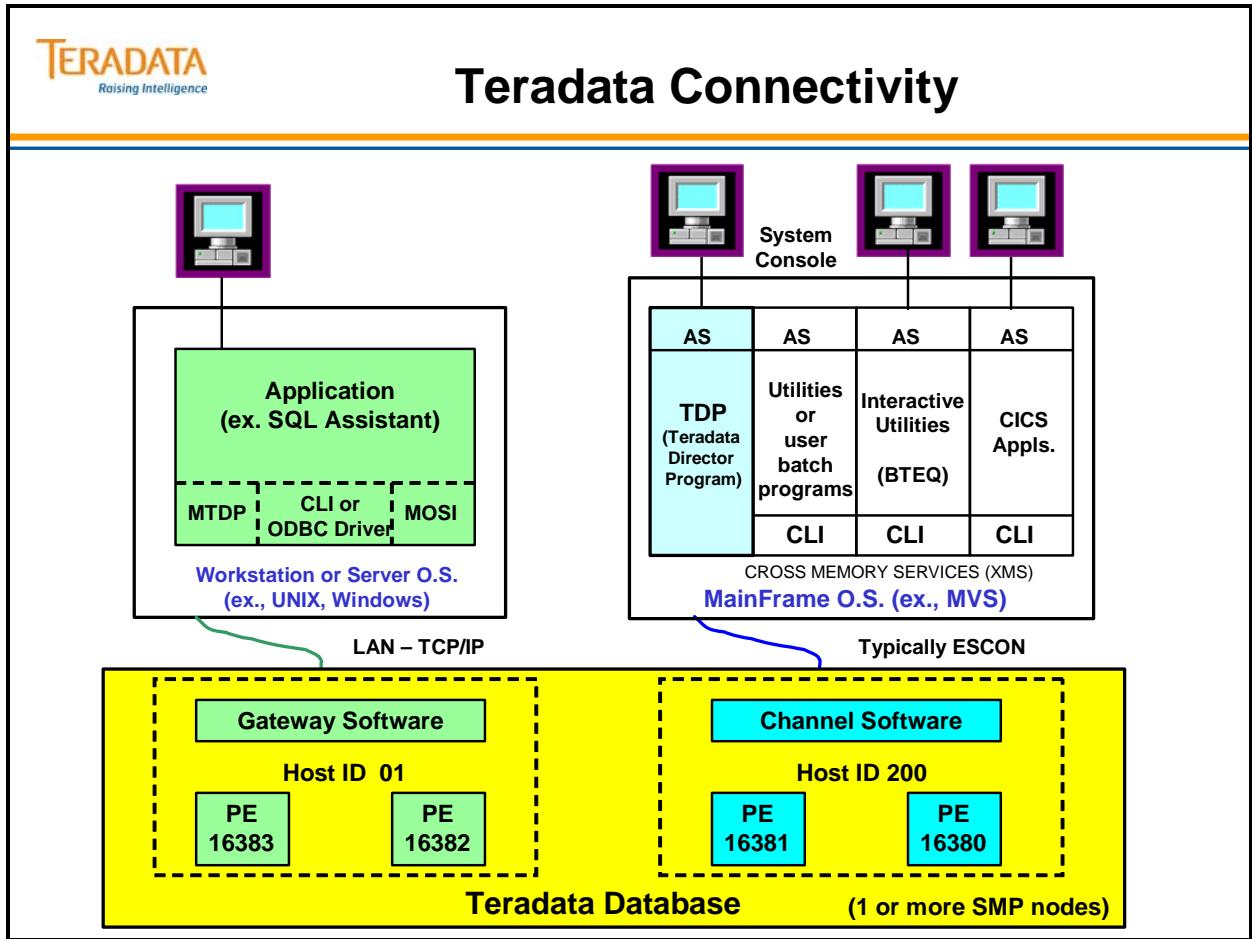
In a local area network (LAN) environment, each workstation on the network will have the utilities and programs needed to access the Teradata database. A network interface card connects workstations directly to the LAN. An Ethernet card in a PCI slot within the processing node connects the node directly to the LAN. These connections provide the workstation operating system access to the gateway software in the node.

The gateway software runs on the Teradata server that is running the Teradata Database. Client programs that communicate through the gateway to the Teradata Database may be resident on the system, or may be installed and running on network-attached workstations.

When a system is configured, it is possible to assign different hostids to different LANs (Ethernet connections) coming into a system. By having multiple hostids (for LANs), a customer can enable/disable a specific LAN. An example might be that you have east/west coast users on different LANs. You can disable the west coast users as a group. If you have multiple LAN hostids, you are effectively setting up "multiple" gateways. Gateway software will balance the number of sessions between the PEs assigned to the hostid for the LAN.

Most customers have multiple Ethernet connections across multiple nodes, but only one hostid is assigned to all LAN connections and there is effectively one gateway in the system. Usually the hostid for LANs has a value of 1; older systems often used a value of 52.

Teradata's gateway software supports up to 1200 sessions per node, depending on available system resources. Gateway errors are handled in the same manner as other database errors.



Sessions and Session Pools

Sessions

A session is a logical connection between an application program and the Teradata Database. A session permits a user to send one request to, and receive one response from, the Teradata server at a time. A session can have only one request outstanding at any time. A user may communicate through one or more active sessions concurrently.

A session is explicitly logged on and off. It is established when the Teradata server accepts the user name and password of the user. When a session is logged off, the system discards the user name and password and does not accept additional Teradata SQL statements from that session.

A session number and a logical client number identify each session to the MVS or VM client. 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.

Session Pools

A session pool is a number of sessions that are logged on to the Teradata server as a unit using the same logon string. 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 is assigned to a specific PE and stays with that PE until the pool ends.

Benefits to session pools include:

- Logons are typically 2-3 seconds faster
- The number of sessions a user can log on is controlled.



Sessions and Session Pools

A user that successfully logs on to Teradata establishes a **session** with Teradata.

A session is:

- A logical connection between the user and the database that permits a user one request and one response at a time.
- Sessions are explicitly logged on to and off from the database and are identified by a logical client ID and a session number.

A session pool is:

- Feature only available with the Teradata Director Program (TDP).
- A number of sessions using the same logon string that are logged on to the Teradata database using a START POOL command (TDP Operator command).
- Reduces the connect time for a mainframe user because a session (with that logon string) has already been established.
 - Logons are typically 2-3 seconds faster.
 - Number of sessions a user can log on is controlled.
- When you utilize a session pool, the TDP does not notify the database when an application/user logs off. It marks the session, “not in use”, and makes it available to another application/user.

Teradata Director Program (TDP)

The Teradata Director Program (TDP) resides in the client mainframe and manages communication between the client application programs and the Teradata server. Key functions of the TDP are listed on the facing page.

To access the Teradata database from a mainframe client, the user makes a request that a Teradata utility or program processes. These requests are directed to a Teradata Director Program (TDP) that resides on the mainframe.

The application talks to Call Level Interface (CLI) which builds the request into a parcel that the TDP sends through the Channel to the PE. When a PE receives a request, the PE formulates the steps to respond to the request and establishes a session with the Teradata database.

The PE sends the processing steps to one or more AMPs where the information is gathered. This information could be the response to a SELECT statement, or it could be a status indicating an INSERT or UPDATE statement was successful.

Communicating with the TDP

The TDP accepts operator commands from the MVS console, MVS/TSO users, VM console, VM/CMS virtual machines, and CLIV2 applications.

Commands you enter from the console are not executed until you execute the RUN command. Messages that result from executing operator commands entered from a console are returned to the console.

Use the MVS MODIFY command from the MVS console to issue TDP operator commands to a TDP already running the MVS environment. The syntax for the MVS Modify command is:

F Tdpid, TDProutine

In the syntax example above, F is the abbreviation for the MVS MODIFY command and **Tdpid** is the four-character identifier associated with the TDP subsystem (for example TDP0, TDP1, and so on) to which the command **TDProutine** is the syntax of the TDP command.



Teradata Director Program (TDP)

The Teradata Director Program (TDP) manages communication between the mainframe's client application programs and the Teradata server.

Key functions include:

- Session initiation and termination
 - Support of session pools
 - Support of NULL password logons
- Logging, verification, recovery and restart notification for client applications
- Manage physical I/O to and from the Parsing Engines (assigned to the channel)
- Security

The TDP can be customized via user-defined exit routines.

- At specific points, you can enable TDP exits and include user-written routines to perform an alteration of normal processing.

TDP Exits

All messages from a channel-attached client sent to and received from the Teradata database pass through the Teradata Director Program (TDP).

For users with channel-attached systems, you can customize the TDP to perform a user-defined exit routine. Customizing the TDP can assist you in collecting information for:

- Performance analysis
- Functional analysis

At three specific points, you can enable TDP exits and include user-written routines to perform some function or alteration of normal processing. These exits will either all be turned on, or all turned off.

There are three supplied exit points:

TDPLGUX – The User Logon Exit Interface is an exit you can use to process Logon Requests.

TDPUTCE – The TDP User Transaction Collection Exit is an exit you may use to process any request or response that traverses the TDP.

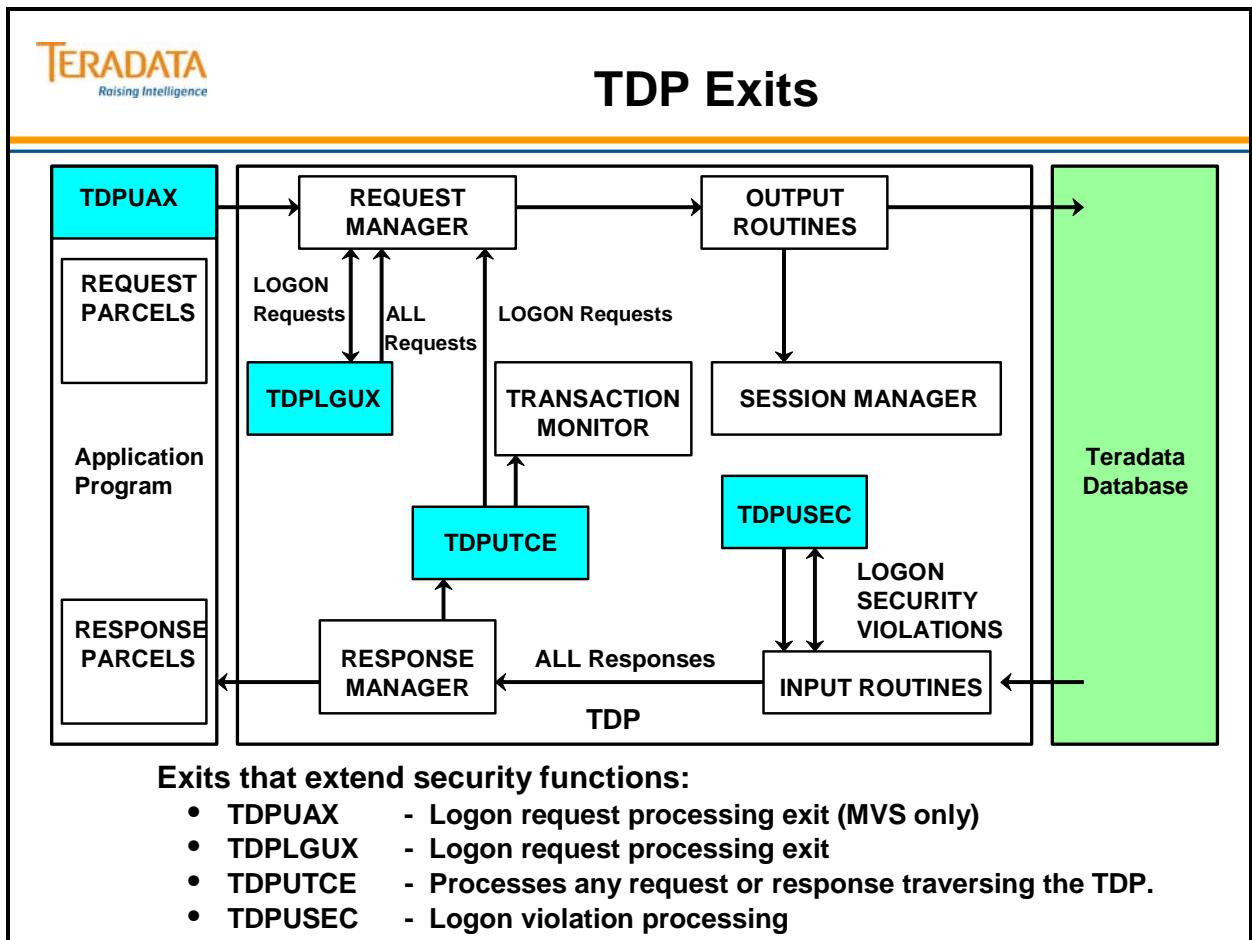
TDPUSEC – The TDP User Security Interface is an exit you use to process logon request denials.

These three exits are available to all TDPs, running on MVS, VM, and OS1100 hosts.

For MVS systems only, an additional exit is provided:

TDPUAUX – The TDP User Address Space exit is called by the TDP when an application initiates a logon or connect request.

Logon example: logon tdpid/userid



Host Logon Processing

The Teradata system default is that any defined user with a valid password who is logged on to a host machine has permission to access the Teradata server through any identified client connection. After installing the software, you may restrict access to the server by associating individual users with specific hosts.

GRANT/REVOKE LOGON Statements

Use the GRANT LOGON statement to give users permission to log on to the Teradata RDBMS from one or more specific client systems. Use the REVOKE LOGON command to retract permission to log on to the Teradata database from one or more specific client systems. These two commands store rows in the DBC.LogonRuleTbl.

You must have EXECUTE privileges on the macro DBC.LogonRule to execute either of these commands.

After installation, use the REVOKE LOGON statement to change the system default by first removing access privileges from all users from all hosts. Then, you can submit the GRANT LOGON statement to assign individual users to specific host IDs.

You can execute the GRANT or REVOKE LOGON statements any time after installation to add or remove user names on individual host connections as needed.



Host Logon Processing

The default is that any authorized user can access Teradata through any identified client connection only if they provide a valid password.

Optionally, an administrator can ...

- grant or deny users permission to logon to Teradata from specific client connections.
- give users permission to logon to Teradata from specific host connections using a NULL password.

The following statements are used to control access from specific “host ids”.

- **GRANT LOGON statement**
 - Gives users permission to logon to Teradata from specific client connections and optionally use a pre-validated logon request.
- **REVOKE LOGON statement**
 - Denies users permission to logon to Teradata from client system(s).

Objects used in Host Logon Processing

You must have EXECUTE privileges on the macro DBC.LogonRule to execute the GRANT LOGON and REVOKE LOGON statements.

Note: DBC.LogonRule is a “dummy macro”. It only has a ; in it.

The GRANT LOGON and REVOKE LOGON statements store rows in the DBC.LogonRuleTbl.

To view the rows in this table use the DBC.LogonRules view.



Objects used in Host Logon Processing

Users who are granted the EXECUTE permission on the following macro can use the GRANT LOGON and REVOKE LOGON statements.

Example:

```
GRANT EXECUTE ON DBC.LogonRule  
TO Sysdba;
```

This allows “Sysdba” to execute the GRANT LOGON and REVOKE LOGON statements.

Execution of **GRANT LOGON or REVOKE LOGON** statements causes rows (representing the rules) to be added or updated in ...

To view the rules in this table, **SELECT** from this view.

DD/D Macro

DBC.LogonRule

DD/D Table

DBC.LogonRuleTbl

DD/D View

DBC.LogonRules[V]

GRANT/REVOKE LOGON Statements

GRANT LOGON and REVOKE LOGON are flagged as non-ANSI when the SQL Flagger is enabled.

Keywords

Keywords you can use with the GRANT and REVOKE LOGON commands include:

HostID	Identifies a mainframe channel connection or a local area network connection that is currently defined to the Teradata RDBMS by the hardware configuration data. The host ID for the Teradata database console is zero (0). For any other connector, the host ID is a value from 1 to 1023.
ALL	The ALL keyword, used in place of a host ID, applies to any source through which a logon is attempted, including the Teradata database console. This is shown as host ID 1024.
AS DEFAULT	Specifies that the current default for the specified host ID(s) is to be changed as defined in this GRANT LOGON statement. A statement with AS DEFAULT has no effect on the access granted to or revoked from particular user names.
TO or FROM username(s)	Overrides the current default for the specified username(s) on the specified host ID(s). The name DBC cannot be specified as a username in a GRANT LOGON statement. A statement that includes this name will return an error message.
WITH NULL PASSWORD	The initial Teradata database default is that all logon requests must include a password. The WITH NULL PASSWORD option, in conjunction with a TDP security exit procedure, permits a logon string that has no password to be accepted on a Teradata system.



GRANT/REVOKE LOGON Statements

```

GRANT LOGON — ON [ host_ID , ] ALL AS DEFAULT [ TO [ username ] ] [ FROM [ ] ] WITH NULL PASSWORD [ ; ]

```

```

REVOKE LOGON — ON [ host_ID , ] ALL AS DEFAULT [ TO [ username ] ] [ FROM [ ] ] [ ; ]

```

host_ID Host number from configuration data. The database console is host number "0" (zero). ALL is represented as "1024".

AS DEFAULT Changes the default for the specified host.

username You can specify up to 25, but not "DBC".

**WITH NULL
PASSWORD** When used in conjunction with a TDP exit or with single sign-on in Windows 2000, overrides the system default that a password is required.

To execute a GRANT or REVOKE LOGON statement,
you must hold execute privileges on the DBC.LogonRule macro.

GRANT/REVOKE LOGON Example

The facing page contains an example of using the REVOKE and GRANT LOGON statements.

COP Entries for LAN Connections

CLI clients work a little differently than ODBC clients. Any CLI based utility will dynamically generate a set of “cop” names at the time you try to make a connection. Example of entries in a “hosts” file:

141.206.28.01	SMP001-7	educ1	educcop1
141.206.28.02	SMP001-8	educ2	educcop2
141.206.28.03	SMP001-9	educ3	educcop3
141.206.28.04	SMP001-10	educ4	educcop4

For example, if you have a 4-node system, you can have four entries for the hostid: TDPIDcop1, TDPIDcop2, TDPIDcop3, TDPIDcop4. Where you put these “cop” entries for address resolution is up to you. COP entries for multiple hosts can be placed in the local hosts file OR in the DNS server file. Most people use DNS, since it is a central repository. If you have “cop” entries in the local hosts file AND they are also in the DNS server file, which are used?

The usual order is to first look in the local /etc/hosts file and then look at DNS server files. With UNIX MP-RAS, you can specify the order of resolution in the "/etc/netconfig" file. With Windows, the default order is to first look in the local hosts file and then escalate to the DNS.

Typically, the place to manage these cop entries is definitely the DNS server. When there are changes, it is much easier to do them in one place rather than on every machine that connects to Teradata.

When you specify a hostid in your logon, the first attempt at connection is to establish the size of the COP pool. First it looks for TDPIPcop1, then TDPIPcop2 ... when an attempt for TDPIPcopn+1 fails, the pool is established as n cops. This "cop" pool is only used to do connection balancing. Another reason for a "cop" pool is to help avoid a single point of connection failure. If the user has the host aliases in the local host file, then the DNS server doesn't get involved until copn if one has local name resolution selected before DNS resolution.

ODBC requires you to create a DSN entry that specifies the machine to connect to. When you create the DSN entry you can give the TDPIP and ODBC will resolve the cop names as they exist at that time and cache them in the registry. There is an option on the screen to create the DSN that says do NOT resolve. In that case, ODBC will behave like CLI by dynamically generating the list of cops to choose from when you make a connection. Something to remember is that most user access to Teradata is via ODBC tools and most DSN entries have those IP addresses cached, so for normal client traffic, there is not a lot of copname resolution that has to be done.



GRANT/REVOKE LOGON Example

GRANT LOGON ON 01 TO tfact08;

Teradata BTEQ 08.02.03.00.
Enter your logon or BTEQ command:
.logon educ/tfact08

.logon educ/tfact08
Password:

*** Logon successfully completed.
*** Transaction Semantics are BTET.
*** Character Set Name is 'ASCII'.
*** Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ ...
.logoff

*** You are now logged off from the DBC.

Notes: This GRANT LOGON creates a specific logon rule in DBC.LogonRuleTbl.
If "REVOKE LOGON ON 01 AS DEFAULT;" is executed, tfact08 can still logon since individual rules override AS DEFAULT.

REVOKE LOGON ON 01 TO tfact09;

Teradata BTEQ 08.02.03.00.
Enter your logon or BTEQ command:
.logon educ/tfact09

.logon educ/tfact09
Password:

*** Error 3026 The user's right to log on has been revoked.
*** Error: Logon failed!

*** Total elapsed time was 3 seconds.

Teradata BTEQ 08.02.03.00.
Enter your logon or BTEQ command:

Notes: This REVOKE LOGON creates a specific logon rule in DBC.LogonRuleTbl.
A "GRANT LOGON ON 01 TO tfact09;" can be executed to allow tfact09 to logon.

Session Related Views

There are three system views that you can use to monitor database access. They are:

DBC.LogonRules	Retrieves information about logon rules generated as a result of successfully processed GRANT/REVOKE LOGON statements. This view uses columns from the DBC.LogonRuleTbl.
DBC.LogOnOff	Supplies information about logon and logoff activity. This view uses columns from the DBC.EventLog table, which records both successful and unsuccessful logon attempts.
DBC.SessionInfo	Provides information about users who are currently logged on. This view uses columns from DBC.SessionTbl.

Dictionary Tables accessed include:

- DBC.LogonRuleTbl
- DBC.EventLog
- DBC.SessionTbl



Session Related Views

DBC.LogonRules[V]

Returns a list of logon rules generated by GRANT and REVOKE statements.

DBC.LogOnOff[V][X]

Provides information about logon attempts (successful or unsuccessful) and logoffs.

DBC.SessionInfo[V][X]

Provides information about the current user or all users currently logged on.

DBC.LogonRules View

The LogonRules view retrieves information about logon rules generated as a result of successfully processed GRANT LOGON statements. This information is stored as rows in the system table DBC.LogonRuleTbl.

This view returns information about the defined rules that you, as the administrator, specify with the GRANT LOGON statement. This statement controls access to the Teradata Database from any server or host.

System administrators or security administrators must specifically authorize user logon requests without passwords.

Example

The SQL statement on the facing page requests a list of the logon rules sorted by username. The response displays that user “tfact06” cannot log on using host ID 200. The users “tfact05 and tfact07” can log on to the database without a password.



DBC.LogonRules View

Provides information about logon rules that are created by GRANT LOGON and REVOKE LOGON statements.

Returns rules from the DBC.LogonRuleTbl.

DBC.LogonRules[V]

UserName	LogicalHostID	LogonStatus
NullPassword	CreatorName	CreateTimeStamp

Example:

List logon rules for
TFACT users.

```
SELECT      *
FROM        DBC.LogonRules
WHERE       UserName LIKE 'tfact%'
ORDER BY   UserName ;
```

Example Results:

Username	LogicalHostID	LogonStatus	NullPassword
tfact05	1024	G	T
tfact06	200	R	F
tfact07	200	G	T
tfact08	1	G	F
tfact09	1	R	F

DBC.LogOnOff View

The DBC.LogOnOff[V][X] views provide information about users who have logged on and off. You can also use this view when you need to know about a user's failed attempts to logon. The facing page shows an example of the DBC.LogonOff view.

DBC.LogOnOff event column definitions include:

	Event	Result
Logon	Bad User Bad Password Bad Account	Logon failed.
Logoff	Forced Off	User had their session aborted.



DBC.LogOnOff View

Provides information about logon and logoff activity, including bad logon attempts and sessions forced off.

DBC.LogOnOff[V][X]

LogDate	LogTime	UserName	AccountName
Event	LogicalHostId	IFPNo	SessionNo
LogonDate	LogonTime	LogonSource	

Example:

List “bad” logon attempts and sessions forced off during the last seven days.

```
SELECT      CAST (LogDate      AS FORMAT 'YYYY-MM-DD')
            ,LogTime
            ,CAST (UserName    AS FORMAT 'X(12)')
            ,Event
FROM        DBC.LogOnOff
WHERE       (Event LIKE 'Bad%'
            OR
            Event LIKE 'Forced%')
AND         LogDate > CURRENT_DATE - 7
ORDER BY    LogDate, LogTime ;
```

Example Results:

LogDate	LogTime	UserName	Event
2008-01-23	10:02:54.84	TFAT30	Bad User
2008-01-23	10:05:22.14	TFACT30	Bad Password
2008-01-23	12:10:13:11	TFACT22	Bad Account
2008-01-24	08:14:11:71	TFACT09	Forced Off

DBC.SessionInfo View

The facing page shows an example of the DBC.SessionInfo[V][X] view.

This view provides information about users who are currently logged on the system. You can use the [X] option of this view to obtain information about the current user.

Example

LogonSource May contain up to 11 fields, depending on the values returned.

Operating system name (e.g., VM or MVS) followed by:

TDP name

VM user ID or MVS job name

Environment name (e.g., TSO, CICS) etc.

Transaction mode:

T = TDBS

A = ANSI

Two PC mode:

2 = 2PC mode

N = Non-2PC mode

Partition

- DBC/SQL = an SQL session
- EXPORT = a FASTEXPORT session
- FASTLOAD = a FASTLOAD session
- HUTPARSE = an ARC data session
- MLOAD = a MULTILOAD session
- MONITOR = sessions running in a performance monitoring application
- NONE = session is recognized but not yet assigned

To get a count of load jobs that are currently executing, you can use the following SQL.

```
SELECT COUNT(DISTINCT LogonSequenceNo) AS Utility_Cnt
FROM DBC.SessionInfo
WHERE Partition IN ('Fastload', 'Export', 'MLoad');
```

The LogonAcct and AccountName columns will usually have the same account id for a user. If a user changes the account id within a session, the AccountName column will reflect the current account id and the LogonAcct will have the logon (or initial) account id.

Both the LogonAcct and AccountName columns will have the actual logon account id (e.g., '\$M_9038_&D&H'), not an expanded account id.



DBC.SessionInfo View

Returns information about the current users or all users currently logged on.

DBC.SessionInfo[V][X]

UserName	AccountName	SessionNo	DefaultDataBase
IFPNo	Partition	LogicalHostId	HostNo
CurrentCollation	LogonDate	LogonTime	LogonSequenceNo
LogonSource	ExpiredPassword	TwoPCMode	Transaction_Mode
ProfileName	CurrentRole	LogonAcct	LDAP
AuditTrailID	CurlolationLevel (12.0)	QueryBand (12.0)	

Example:

List all users currently logged on, the source of their session, the date they logged on, and their connect time.

```
SELECT      TRIM (UserName) ||' From ' || LogonSource
            (FORMAT 'X(22)')      AS Logons
            ,LogonDate
            ,TIME - LogonTime
            (FORMAT '99:99:99')  AS ConnectTime
FROM        DBC.SessionInfo
ORDER BY    UserName ;
```

Example Results:

Logons	LogonDate	ConnectTime
TFACT01 From (TCP/IP)	2008-02-10	01:42:11
TFACT02 From (TCP/IP)	2008-02-10	00:09:02
TFACT04 From (TCP/IP)	2008-02-10	06:53:19
TFACT05 From (TCP/IP)	2008-02-10	02:05:12
TFACT05 From (TCP/IP)	2008-02-10	00:16:15

Additional Utilities to View Sessions

Additional tools that may be used to view session activity are listed on the facing page.

Gateway Global Utility

Gateway Global is not as commonly used as it once was because the Sessions display of Teradata Manager is easier to use. However, you can still access the Gateway Global Utility by invoking the following commands:

Command-line version: ***gtwglobal***
Command for X-version: ***xgtwglobal***

Session Control

The Gateway Global utility allows you to monitor and control Teradata database network-attached users and their sessions. For example, by starting the utility and issuing utility commands with this utility, you can monitor network sessions and traffic, disable logons, force users off the Teradata database and diagnose gateway problems.

Disconnect and Kill Commands

The Disconnect User/Session and Kill User/Session commands are similar in that they both disconnect sessions from the database. The Kill command will abort one session immediately or all sessions of a particular user, then log the user off. The Disconnect command simply puts the sessions in a disconnect state and does not log the user off. The database is still aware of the sessions, and if the user re-establishes the connection from their client workstation, the sessions are allowed to re-connect.

Examples of Gateway Global Commands

Network and Session Information

DISPLAY NETWORK	Displays your network configuration.
DISPLAY GTWALL	Displays all sessions connected to the gateway.
DISPLAY SESSION	Displays information about a specific session on the gateway.

Administering Users and Sessions

DISABLE LOGONS	Disable logons to the RDBMS through the gateway.
ENABLE LOGONS	Enable logons to the RDBMS 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.



Additional Utilities to View Sessions

Teradata Manager and Performance Monitor – Windows utilities

- Both of these utilities provide functions to view and abort sessions
- Can be used to change a session's priority
- Performance Monitor may be executed independently or via Teradata Manager
 - Details on Teradata Manager and Performance Monitor will be covered later

QrySessn – system utility started via Database Console or Teradata Manager

- Provides display of active and idle sessions
- Supervisor can be used to abort sessions

gtwglobal – system utility

- This utility can be used to monitor/abort only gateway or LAN-based sessions
 - gtwglobal – command-line utility version which can be used with Windows or UNIX
 - xgtwglobal – X windows (UNIX) version of this utility
- Can be used to view and abort sessions
- Not as commonly used because Teradata Manager provides an easier interface

Teradata Manager Sessions

An example of the **Sessions** display from Teradata Manager is shown on the facing page.

From the **Teradata Manager** menu, click **Monitor > Sessions**, and choose the filter for the types of sessions to view:

- **All** - shows all sessions currently on the Database
- **Active** - shows all active sessions
- **Blocked** - shows sessions that are blocked by other sessions
- **Idle** - lists information about inactive and idle sessions
- **Parsing** - lists information about parsing sessions
- **Responding** - lists information about responding sessions
- **Aborting** - shows sessions that are in the process of aborting
- **Other** - lists information about sessions when there is a difference in the state of the AMP and PE or if the state is not Idle, Active, Blocked, Parsing, Responding, Aborting or Delayed, including those that are currently logged on to the Monitor Partition.
- **Delayed** - shows sessions that are delayed

To view session details, either double-click the session number, or right-click the number of the session to display the shortcut menu, and click **Session Details**.

Other options include:

- **Modify Session Priority** – modify session's account to change priority.
Note: Modifying accounts is allowed only in the DBC/SQL partition; therefore, this option is enabled only when the partition is DBC/SQL.
- **Abort Session**- abort this session
- **Blocked By** - display a report showing which sessions (if any) are blocking this session
- **Blocking** - display a report showing which sessions this one is blocking
- **Current SQL** - view the SQL statements currently being executed by this session, along with job step information and associated Explain text
- **Skew** - display a report showing skew (workload imbalance) for this session
- **Modify Session Workload** - modify session workload settings
- **Release Request** - release a request that is on hold
- **Query Band** - display a report showing the query band pairs applied to this session

TERADATA
Raising Intelligence

Teradata Manager Sessions

The Sessions display of Teradata Manager allows you to view and control sessions.

To access this display:

Monitor Menu > Sessions

This pull down menu can be accessed by right-clicking on the session number.

The screenshot shows the Teradata Manager Sessions window. The title bar reads "Teradata Manager (dbcmanager) [profile DEFAULT@tdt5-1] - [ALL Sessions (tdt5-1)]". The main area displays a table of sessions with columns: Session, User Name, Account, Pri, Request Count, State, AMP CPU, delta CPU, CPU Skew, AMP I/O, and D. There are 10 sessions listed. A context menu is open over session 03391, listing options: Session Details, Modify Session Priority, Abort Session, Blocked by, Blocking, Current SQL, Skew, Modify Session Workload, Release Request, and Query Band. The "Abort Session" option is highlighted.

Session	User Name	Account	Pri	Request Count	State	AMP CPU	delta CPU	CPU Skew	AMP I/O	D
03380	TLJC08	\$M_TT_&S_&D&H	\$M	20	IDLE	17	0	0	20684	
03382	TLJC06	\$M_TT_&S_&D&H	\$M	49	IDLE	10	0	0	16968	
03391	TLJC04_A	\$M	\$M	4	IDLE	3	0	0	4059	
03391	TLJC04_A	\$M	\$M	23	ACTIVE	1	0	0	924	
	Session Details	\$M	\$M	38	IDLE	57	0	0	60306	
	Modify Session Priority	\$M	\$M	1	IDLE	0	0	0	0	
	Abort Session	\$M	\$M	22	IDLE	2	0	0	3588	
	Blocked by	\$C-MANAGER	\$H	7	ACTIVE	0	0	8	0	
	Blocking	\$H	\$M	20	ABORTING	3	0	7	2953	
	Current SQL	\$M	\$M	5	IDLE	0	0	0	214	

Right click on a Session Number for additional information

Query Session Utility

The Query Session utility provides information about all Teradata sessions. To start the utility, enter ***START QRYSESSN*** in the Supervisor window. A Teradata session may be in one of several possible states. They include:

State	Description
Unknown	The session number is not recognized.
Idle	Process is not taking place at this time.
Delay (V2R6)	The query is on a Teradata DWM Delay queue.
Parsing	Session is in the DBC/SQL parser phase.
Active¹	Session has sent steps to the dispatcher/AMPs
Aborting¹	Session is aborting the latest request.
Blocked²	Session is waiting for a database lock to release.
Response	Response to session request is in process.

Archive/Recovery, FastLoad, MultiLoad, FastExport, and other utility status information is also provided.

The Query Session utility can also be started from HUTCNS by entering the following command: **SES** or **ses**

Example

The example on the facing page shows a sample Query Session Report and the headings displayed by Query Session for this report. You can use an asterisk (*) as a wild card symbol to depict all hosts and/or all sessions. If a session is idle, only the session identifier information would be displayed.

The complete prompt for detailed information is ...

“Is detailed information needed for HUT/FASTLOAD/MLOAD/EXPORT? y-yes, n-no

Answering yes to this prompt provides more details for the archive and load utilities.

¹Shows CPU time (all AMPS) in 100ths of a second and total segment access calls.

²Shows if lock was requested and if the lock encountered was a host utility lock (archive).



Query Session Utility

Qrysessn prompts the user for:

Please enter logical host id? 1
 Please enter session ids? *
 Is detail information needed? y

From the Supervisor, you can also abort sessions.

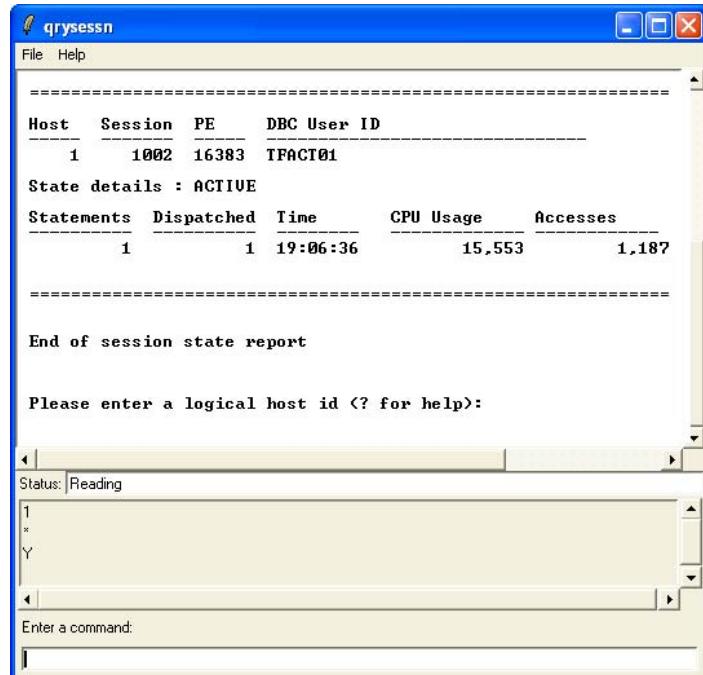
The ABORT SESSION command aborts the currently running SQL transaction in progress.

The LOGOFF option also terminates the user session.

ABORT SESSION hostid:ses# [LOGOFF]
 hostid.username
 *.username
 hostid.*
 **

Example:

ABORT SESSION 1:1002 LOGOFF



From Supervisor, enter: START QRYSESSN

Access Control Mechanisms

You can control user access by granting access to specific views and macros. 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.

User Privileges

During a session, the Teradata Database system accesses the user's default database to search for or store the occurrence of an object whose reference in the SQL statement is not qualified with a database name.

The user can override the default for a particular object by qualifying statement references with a database name (in the form `databasename.objectname`).

At any time during the session, the user can override the current default by executing the SQL DATABASE statement. The system uses the space associated with the specified or default database as the default until the user executes another DATABASE statement or logs off.

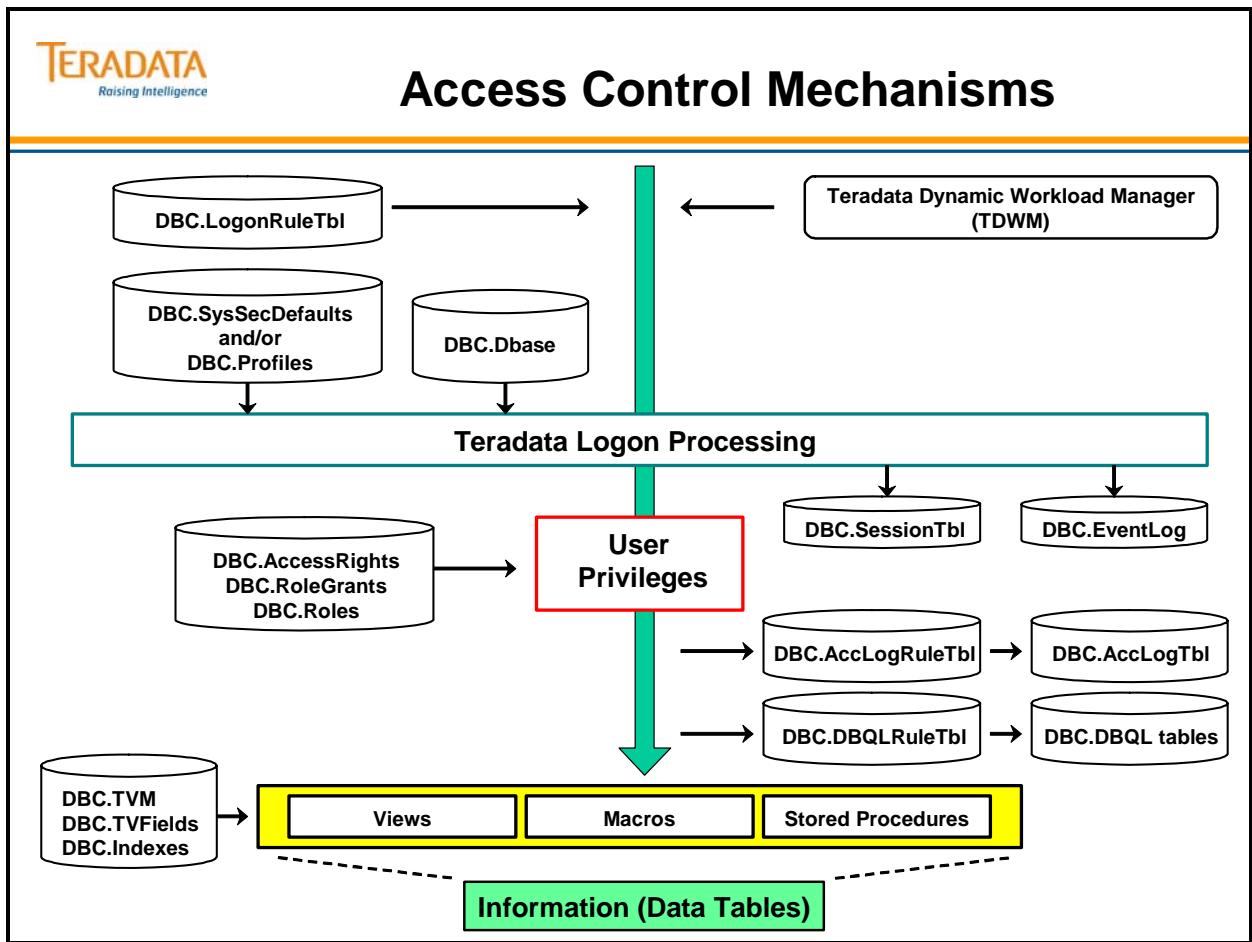
An arrangement of predefined privileges or access rights control the user's activities during a session. Access rights are associated with a user, a database, and an object (table, view, or macro).

The system verifies a user's access rights when the user attempts to access or execute a function that accesses an object. Teradata stores access rights information in the system table DBC.AccessRights. You can retrieve this information by querying the DBC.UserRights view.

As the administrator, there are two additional methods you can use to limit user access to the Teradata Database:

- Create views
- Create macros and/or stored procedures

The facing page shows a diagram of access control mechanisms in Teradata.



Using Views to Limit Access

Another method of controlling user access to data is through the structure of views and macros. Views limit access to table columns or rows that contain sensitive information. Macros limit those actions a user can perform on table columns or rows.

To view the underlying tables of a view and order the nesting of a view, use the following syntax.

SHOW SELECT * FROM viewname;

Example

The example on the facing page demonstrates how to create a view that limits user access to data.

An existing table called Employee contains some sensitive information. As the administrator, you need to create a view that allows users to see only certain information. You create a new view called Staff201 as shown on the following page.

After you create the view, you must GRANT SELECT privileges to all users that need to access the new restricted view. If a view is to be used to UPDATE values in a base table, the prospective users must also be given the UPDATE privilege.

Users submit the SELECT statement to access the new restricted view. The user only sees selected rows and columns as if they were looking at a complete table. The user does not know that the underlying table contains more information. The user does not realize he or she is looking at a restricted view.

With **CHECK Option**

When the WITH CHECK OPTION is ...	Any INSERT or UPDATE made through the view ...
used	only creates or modifies rows that satisfy the WHERE clause
not used	Ignores the WHERE clause used in the definition of the view.



Using Views to Limit Access

Employee

Employee Number	Manager Employee Number	Department Number	Job Code	Last Name	First Name	Hire Date	Birth Date	Salary Amount
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	861015	631015	3945000
1008	1019	301	312102	Kanieski	Carol	870201	680517	3925000
1005	0801	403	431100	Ryan	Loretta	861015	650910	4120000
1004	1003	401	412101	Johnson	Darlene	861015	560423	4630000
1007	1005	403	432101	Villegas	Armando	870102	470131	5970000
1013	0801	401	411100	Trader	James	860731	570619	4785000

```
CREATE VIEW Payroll_VMDB.Staff_401 AS
```

```
SELECT
    Employee_Number AS EmpNo
    ,Last_Name      (FORMAT 'X(10)')
    ,First_Name     (FORMAT 'X(10)')
    ,Hire_Date      (FORMAT 'YYYY-MM-DD')
    ,Salary_Amount  (FORMAT '$ZZZ,ZZ9.99')
FROM    Payroll.Employee
WHERE   Department_Number = 401
AND    Salary_Amount < 50000
WITH CHECK OPTION;
```

```
GRANT SELECT ON Payroll_VMDB.Staff_401
    TO user_1;
```

```
GRANT UPDATE (Last_Name, Salary) ON
    Payroll_VMDB.Staff_401 TO user_1 ;
```

user_1 executes the following SQL:

```
SELECT      *
FROM        Staff_401
ORDER BY    Salary DESC;          (success)

UPDATE      Staff_401
SET         Salary_Amount = 58000
WHERE       EmpNo = 1013;         (fails)
```

Note: Failure 3564 Range constraint;
Check error in field Employee.Salary_Amount.

Using Macros to Reduce User SQL Complexity

Teradata includes referential integrity. As DBA, you can create macros to control access and to optionally ensure referential integrity and reduce end user complexity.



Using Macros to Reduce User SQL Complexity

```

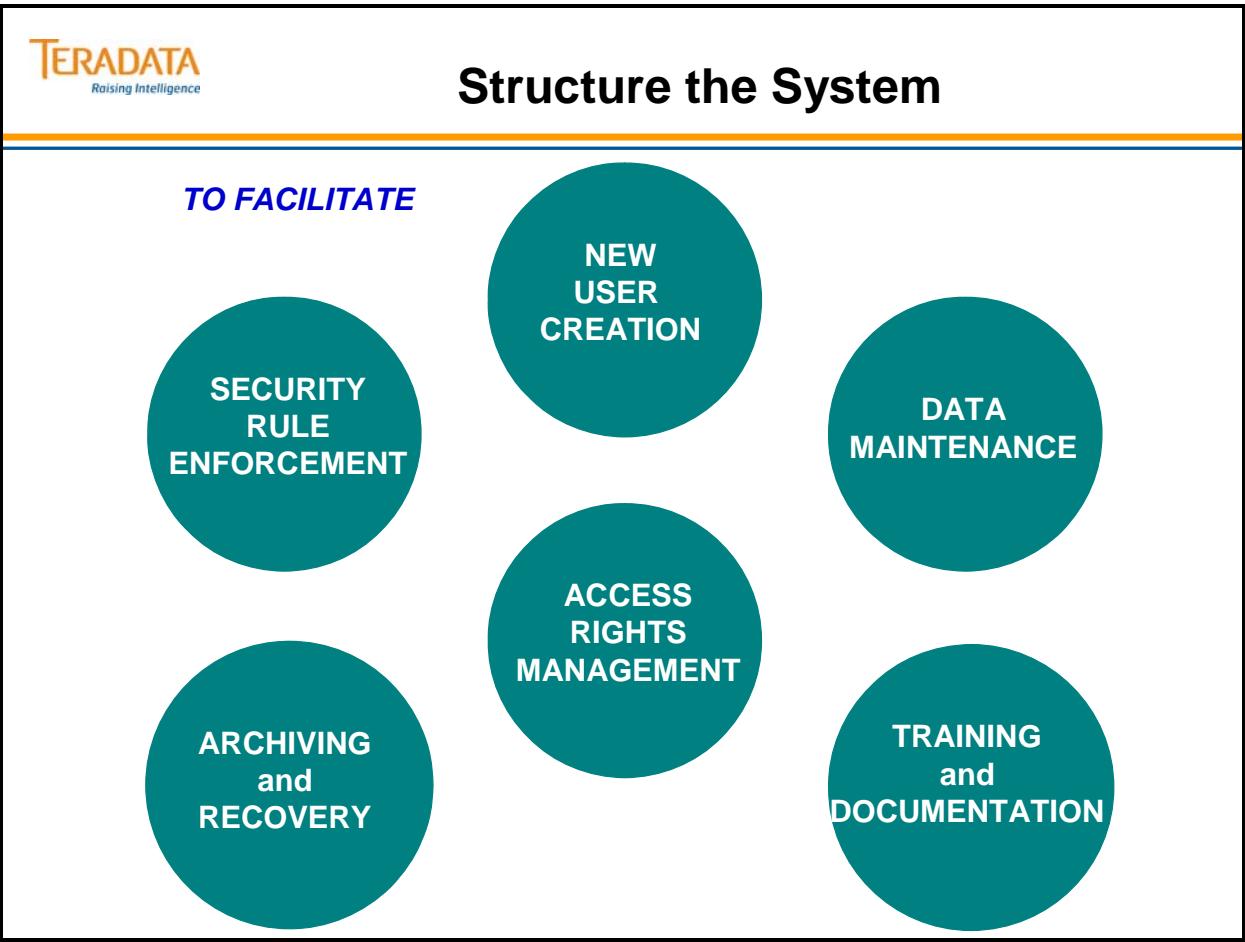
CREATE MACRO new_employee
  ( number          INTEGER
    ,mgr            INTEGER
    ,dept           INTEGER
    ,job            INTEGER
    ,lastname       CHAR (20)
    ,firstname      VARCHAR (30)
    ,hired          DATE
    ,birth          DATE
    ,salary         DECIMAL (10,2))
AS
  ( ROLLBACK 'Invalid hire'      WHERE (:hired - :birth) / 365 < 21 ;
  ROLLBACK 'Invalid Department' WHERE :dept      NOT IN
    (SELECT department_number FROM Department WHERE department_number = :dept) ;
  ROLLBACK 'Invalid Job Code'   WHERE :job      NOT IN
    (SELECT job_code        FROM Job      WHERE job_code = :job);
  INSERT INTO Employee
    ( employee_number
      ,manager_employee_number
      ,department_number
      ,job_code
      ,last_name
      ,first_name
      ,hire_date
      ,birthdate
      ,salary_amount )
  VALUES ( :number, :mgr, :dept, :job, :lastname, :firstname, :hired, :birth, :salary ) ; )
EXEC new_employee (748102, 321912, 2133, 8309, 'Nomar', 'Joseph', 890512, 450824, 49038.00);

```

Structure the System

As the administrator, it is your responsibility to manage access rights. Managing access rights is important for:

- New user creation
- Security rule enforcement
- Data maintenance
- Training and documentation
- Archiving and recovery



A Recommended Access Rights Structure

An access rights structure recommended for the Teradata database includes has the following characteristics:

- All users belong to a database and inherit their access rights.
- Users do not have direct access to data tables, unless they are performing batch operations.
- Users access databases that contain only views and macros.
- VMDB databases contain *only* views and macros.
- TABLE databases contain *only* tables.
- Access rights are *only* extended at the database or user level, not at the individual table level.

Example

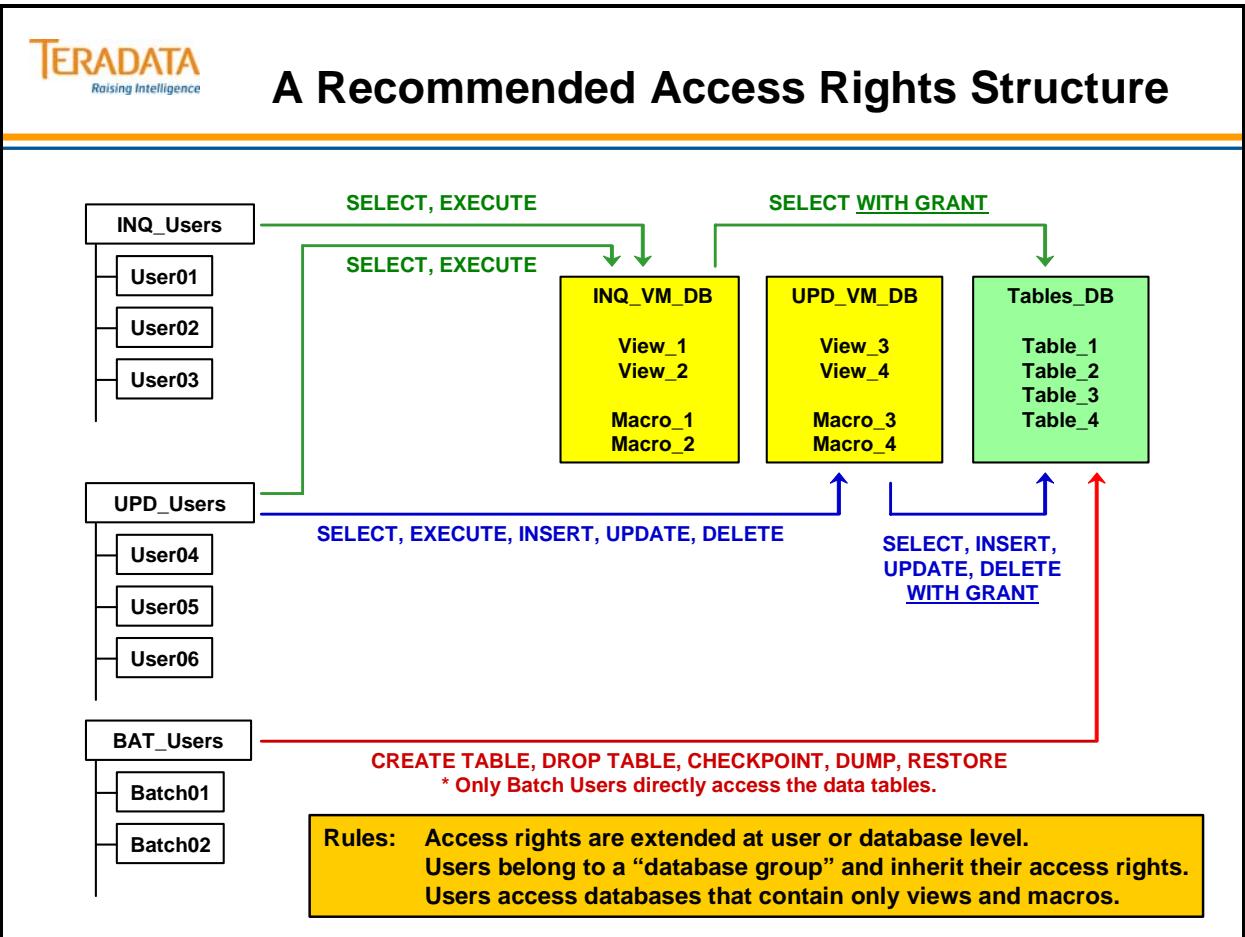
The diagram on the facing page illustrates an example of the suggested Teradata access rights scheme. This scheme has three user databases:

INQ_Users	Users that belong to the Inquiry database inherit SELECT and EXECUTE privileges when you create them.
UPD_Users	Users that belong to the Update database inherit SELECT, EXECUTE, INSERT, DELETE and UPDATE privileges when you create them.
BAT_Users	Users that belong to the Batch database inherit DROP and CREATE TABLE, CHECKPOINT, DUMP, and RESTORE privileges when you create them.

In addition to the access rights stored in each user database, the Inquiry VM and Update VM databases also contain a set of access rights. Both are discussed below:

INQ_VM_DB	The Inquiry VM Database contains views and macros that give Inquiry users access to information. The database has the SELECT privilege with GRANT OPTION.
UPD_VM_DB	The Update VM Database contains views and macros that enable Update users to modify information. This database has the SELECT, INSERT, DELETE and UPDATE privileges with GRANT OPTION.

The WITH GRANT option enables the Upd_VM_Database to give the necessary privileges to the update users.



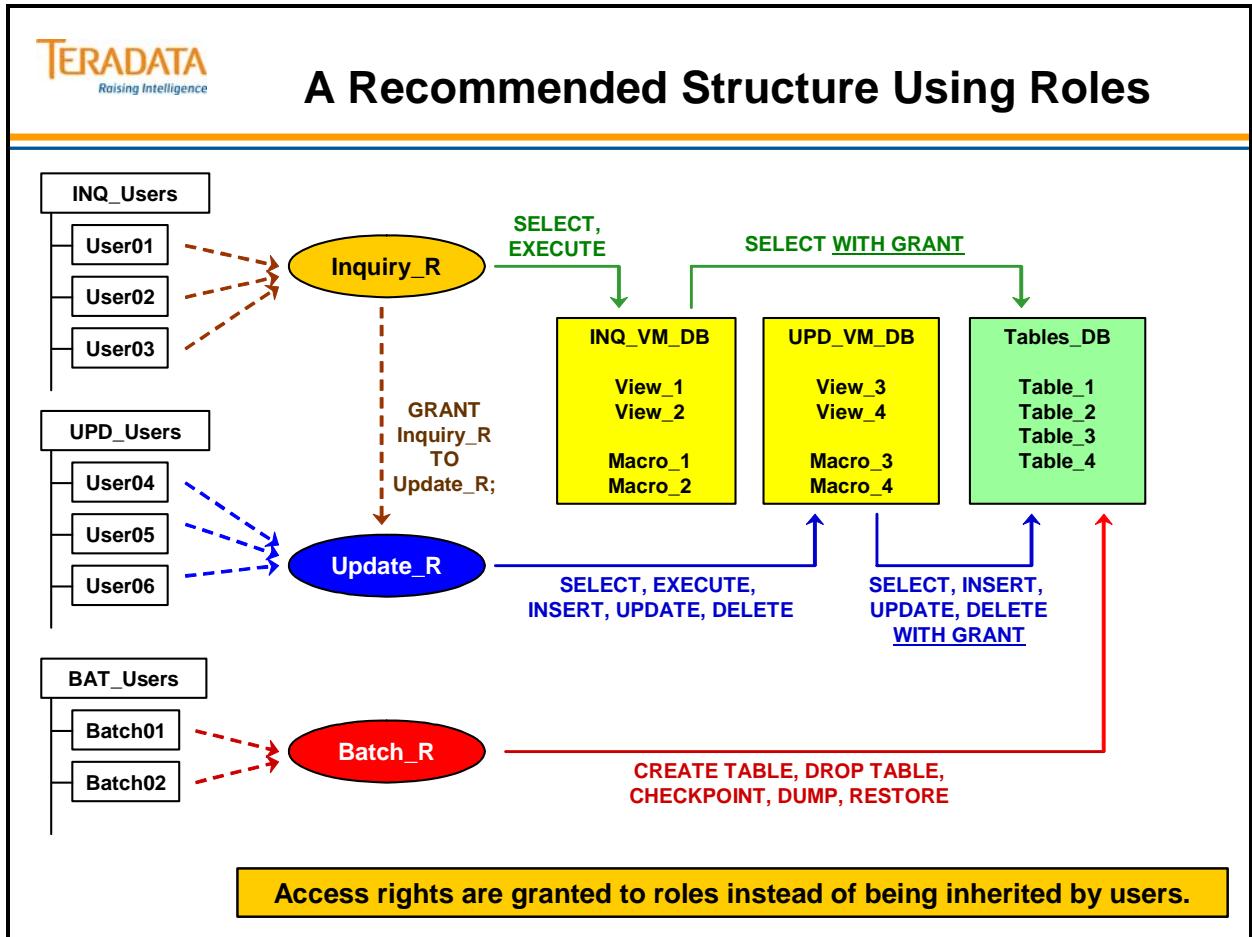
A Recommended Structure Using Roles

The example on the facing page illustrates an example of the suggested Teradata access rights scheme utilizing roles. This scheme has three roles:

Inquiry_R	Users granted to the Inquiry role get SELECT and EXECUTE privileges associated with this role.
Update_R	Users granted to the Update role get SELECT, EXECUTE, INSERT, DELETE and UPDATE privileges associated with this role.
Batch_R	Users granted to the Batch role get DROP and CREATE TABLE, CHECKPOINT, DUMP, and RESTORE privileges associated with this role.

In addition to the access rights assigned to the roles, the Inquiry VM and Update VM databases also contain a set of access rights. Both are discussed below:

INQ_VM_DB	The Inquiry VM Database contains views and macros that give Inquiry users access to information. The database has the SELECT privilege with GRANT OPTION.
UPD_VM_DB	The Update VM Database contains views and macros that enable Update users to modify information. This database has the SELECT, INSERT, DELETE and UPDATE privileges with GRANT OPTION.



A Recommended System Hierarchy

A system structure recommended for the Teradata database is shown on the facing page. Each major application function has an associated administrator that would have control of the users and databases within that application function.

Keys to the hierarchy on the facing page are:

INQ_Users – database or set of “inquiry” users

UPD_Users – database or set of “update” users

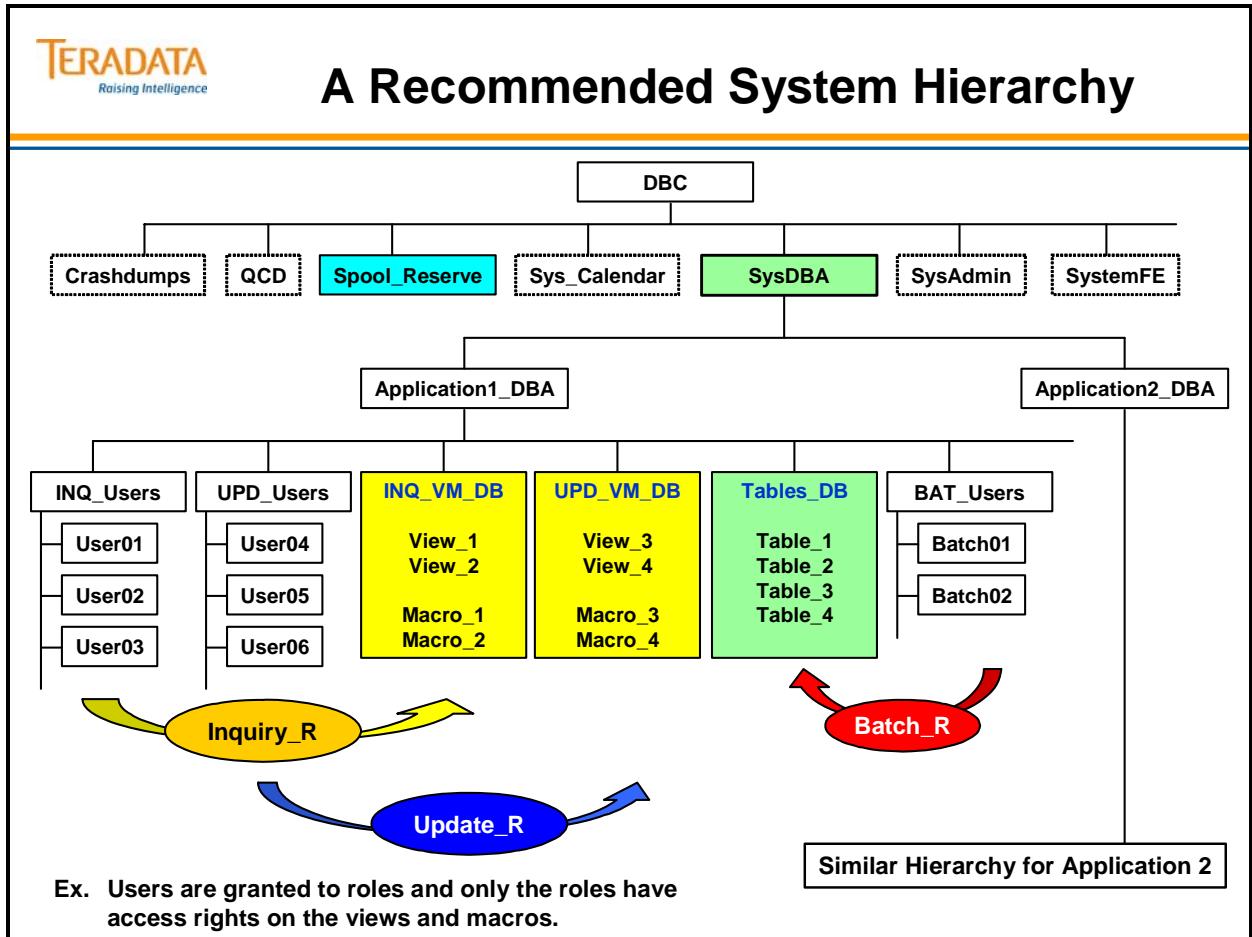
INQ_VM_DB – database of views and macros that access tables in Tables_DB

UPD_VM_DB – database of views and macros that update tables in Tables_DB

Tables_DB – database of user data tables

BAT_Users – database or set of “batch” users; operational users that execute utilities that directly access the tables (e.g., FastLoad)

Optionally roles can be used to easily maintain access rights and reduce the number of access rights.



System Access Controls Summary

The facing page summarizes some important concepts regarding this module.



System Access Controls Summary

- The mission of security administration is to prevent unauthorized user access to the database and its resources.
- To protect system access:
 - Associate passwords with usernames.
 - Associate application users with specific hosts.
- You can control database access by granting access to views and macros.
 - Views can limit access to certain columns and rows.
 - Macros can limit the actions a user can perform.
- Good access rights management facilitates your role as system administrator in security rule enforcement, data maintenance, archive and recovery, and other areas.
- Characteristics of a good database structure include:
 - Users belong to a database group and inherit their access rights.
 - Users do not have direct access to tables.
 - Access rights are extended at the database or user level (not at the individual table level).

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. True or False. Although usernames are the basis for identification in the system, username information usually is not protected information.
2. True or False. Once users have a username and password, they can access any information in the system.
3. True or False. If you want to change the minimum number of characters in a valid password from 4 to 6, you would update the DBC.LogonRules table.
4. What does 1024 represent in the DBC.LogonRules view? _____
5. Which of the following choice(s) can be used to view host or mainframe sessions? _____
 - A. QrySessn utility
 - B. Sessions utility
 - C. DBC.SessionInfo view
 - D. Gtwglobal utility
6. Which one of the following choices is used to determine why a user logon has failed? _____
 - A. DBC.Logons view
 - B. DBC.AccessLog view
 - C. DBC.LogonEvents view
 - D. DBC.SessionInfo view
 - E. DBC.LogOnOff view

Teradata Training

Notes

Module 49



Access and Query Logging

After completing this module, you will be able to:

- **Describe how the BEGIN/END LOGGING statements capture information about user access to Teradata.**
- **Describe how to set up user access logging.**
- **Use system views to gather information about data access.**
- **Identify the reasons for using the Database Query Log.**
- **Identify the tables and views that make up the DBQL facility.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Access and Query Logging	49-4
Access Logging.....	49-6
Objects used in Access Logging	49-8
BEGIN LOGGING Statement	49-10
END LOGGING Statement	49-12
Setting up Access Logging	49-14
Log Entries.....	49-14
Keywords and Object-Names	49-14
Access Log Views.....	49-16
DBC.AccLogRules View.....	49-18
BEGIN LOGGING – Example	49-20
DBC.AccessLog View	49-22
DBC.AccessLog View – Example.....	49-24
END LOGGING – Example	49-26
Teradata Administrator – Tools Menu > Access Logging.....	49-28
Query Logging (DBQL) Concepts.....	49-30
Provides Collection of Historical records based on Rules	49-30
Objects used in Defining Rules for DBQL	49-32
Rules.....	49-32
Objects used in DBQL (cont.).....	49-34
BEGIN QUERY LOGGING Statement.....	49-36
BEGIN QUERY LOGGING.....	49-36
BEGIN QUERY LOGGING WITH ... (cont.).....	49-38
BEGIN QUERY LOGGING LIMIT ... (cont.).....	49-40
BEGIN QUERY LOGGING Examples.....	49-42
BEGIN QUERY LOGGING Examples (cont.)	49-44
BEGIN QUERY LOGGING Examples (cont.)	49-46
END QUERY LOGGING Statement.....	49-48
DBC.DBQLRules View	49-50
DBC.QryLog View – Example	49-52
DBC.QryLogSummary View – Example	49-54
Teradata Administrator – Tools Menu > Query Logging	49-56
Access and Query Logging Summary.....	49-58
Review Questions	49-60
Lab Exercise 49-1	49-62
Lab Exercise 49-1 (cont.).....	49-64
Lab Exercise 49-2	49-66
Lab Exercise 49-2 (cont.).....	49-68
Lab Exercise 49-2 (cont.).....	49-70

Access and Query Logging

Use Access Logging for security and auditing purposes.

Use DBQL to capture details needed for workload analysis, performance tuning, and resource usage analysis.

Access Logging Facility

The Access Logging facility provides an administrator with the capability to monitor data access requests in the system and log granted and/or denied requests.

With Access Logging, logged rows are written immediately to disk before the SQL is executed, which can slow down short query work. At the same time, this approach to logging does offer higher reliability and can register negative accesses (attempts to view data that did not succeed), which would not show up in DBQL.

Query Logging Facility

The Database Query Log (DBQL) is a feature (starting with Teradata V2R5) 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.

DBQL provides a series of predefined tables that can store, based on rules you specify, historical records of queries and their duration, performance, and target activity.

DBQL is flexible enough to log information on the variety of SQL requests that run on Teradata, from short transactions to longer-running analysis and mining queries. You begin and end collection for a user or group of users and/or one or a list of accounts.

DBQL is streamlined for collection efficiency and provides more detail about the query than the Access Log. On the other hand, because DBQL caches data in memory before writing it periodically to disk, there is some delay in seeing the logged data, and it is possible to lose data that is still in the cache on a restart.



Access and Query Logging

There are two logging facilities available to the database and/or security administrator.

- **Access Logging Facility**
 - Used for access and security audit analysis.
 - May be used to monitor data access requests (via access rights checks) and log entries for requests that are granted and/or denied.
- **Query Logging Facility (DBQL)**
 - Used for query activity and workload analysis.
 - Can be used to track processing behavior and/or capture detailed information about the queries that are running on a system.
 - Workloads can be utilized with Teradata Analyst tools such as Teradata Index Wizard.
 - Facility available starting with V2R5.

Access Logging

The Access Logging facility provides an administrator with the capability to monitor data access requests in the system and log granted and/or denied requests.

The DDL statements BEGIN LOGGING and END LOGGING are used to control the monitoring of access rights checks performed by the Teradata Database. Each time you execute a BEGIN LOGGING statement, the system table DBC.AccLogRuleTbl receives applicable rule entries. (The system view DBC.AccLogRules offers access to the contents of this table.)

When a user named in a BEGIN LOGGING statement attempts to execute a specified action against a specified object, the Teradata Database checks the access rights necessary to execute the statement according to the rules in DBC.AccLogRuleTbl. The privilege checks made and/or the access results are logged in the system table DBC.AccLogTbl. (The system view DBC.AccessLog offers access to the contents of this table.)

A logging entry does not indicate that a statement was executed; rather, it indicates that the system checked the privileges necessary to execute the statement.

You can terminate logging by submitting an END LOGGING statement for any action, user, or object for which logging is currently active. Note that you cannot end logging begun for a specific username by omitting the BY *username* option.



Access Logging

An administrator can ...

- use the Access Logging facility to monitor data access requests and log entries for requests that are granted and/or denied.
- optionally capture the SQL text along with the access right check.

The following statements are used to specify objects and/or SQL requests to monitor for specific or all users.

– **BEGIN LOGGING statement**

- Starts the monitoring of data access requests by Teradata.

– **END LOGGING statement**

- Ends the monitoring of data access requests by Teradata.

Objects used in Access Logging

You must have EXECUTE privileges on the macro DBC.AccLogRule to execute the BEGIN LOGGING and END LOGGING statements.

Note: DBC.AccLogRule is a “dummy macro”. It only has a ; in it.

The BEGIN LOGGING and END LOGGING statements start and stop the auditing of data access requests. The BEGIN LOGGING and END LOGGING statements store rows in the DBC.AccLogRuleTbl. To view the rows in this table, use the DBC.AccLogRules[V] views.

If the user does not submit a BEGIN LOGGING statement, then by default the system does not generate any entries on any user action.

When an object or SQL request (identified in one of the access log rules) is accessed, an entry is logged in the DBC.AccLogTbl. To view the rows in this table, use the DBC.AccessLog[V] views.



Objects used in Access Logging

Users who are granted EXECUTE permission on the following macro can use the BEGIN LOGGING and END LOGGING statements.

Example:

GRANT EXECUTE ON DBC.AccLogRule TO SecAdmin;

This allows "SecAdmin" to execute the BEGIN LOGGING and END LOGGING statements.

Execution of **BEGIN LOGGING** or **END LOGGING** statements causes rows (representing the rules) to be added or updated in ...

To view the rules in this table, **SELECT** from this view.

Based on the rules, access of specified objects or SQL statements cause entries to be placed in ...

To view the log of entries, **SELECT** from this view.

DD/D Macro

DBC.AccLogRule

DD/D Table

DBC.AccLogRuleTbl

DD/D View

DBC.AccLogRules[V]

DD/D Table

DBC.AccLogTbl

(can potentially become large)

DD/D View

DBC.AccessLog[V]

BEGIN LOGGING Statement

Teradata verifies a user's access rights when the user attempts to access an object. As the administrator, you can capture information about checks performed on a user's access rights with the BEGIN LOGGING statement.

When you activate logging, the specified privilege check performed by the Teradata Database generates a row in the DBC.AccLogTbl. Later, you can use system-supplied views to monitor and analyze the information stored there.

The BEGIN LOGGING statement has a number of options. Several are described below:

DENIALS — Tracks only those entries when statement execution fails because the user does not have the privilege(s) necessary to execute the statement.

FIRST, LAST, EACH — Defines the frequency with which log entries are made. The default for BEGIN LOGGING is FIRST.

ALL — Tells the system to make a log entry when the user attempts certain actions against the specified object, including:

CD	CREATE DATABASE	DP	DUMP	I	INSERT
CM	CREATE MACRO	D	DELETE	S	RETRIEVE/SELECT
CP	CHECKPOINT	CU	CREATE USER	RS	RESTORE
CT	CREATE TABLE	DU	DROP USER	U	UPDATE
CV	CREATE VIEW	G	GRANT	E	EXECUTE
DD	DROP DATABASE/USER				

BY username — Lists the users for which the system will make log entries. The default is all users.

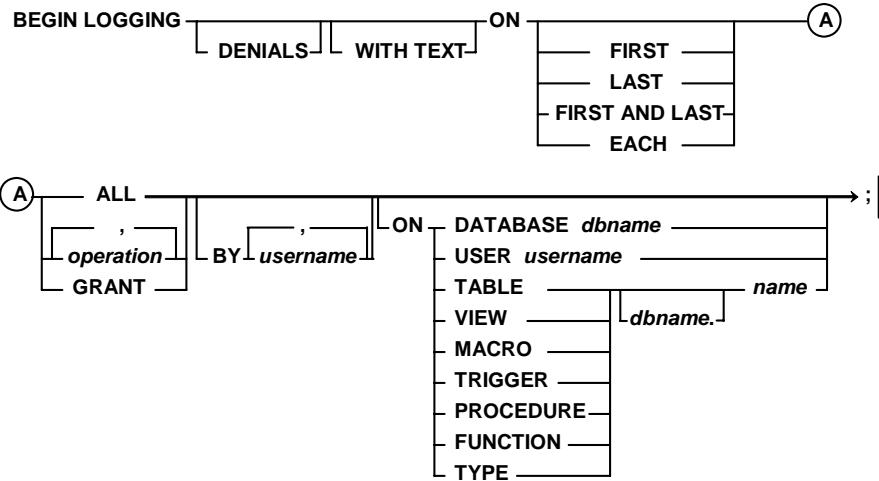
ON keyword object-name — Defines which objects will generate rows in the log table when a user attempts to access them. The keyword object-name combinations must be one of the following:

USER	User name	TRIGGER	Trigger name
DATABASE	Database name	MACRO	Macro name
TABLE	Table name	PROCEDURE	Stored procedure name
VIEW	View name	FUNCTION	User-defined function name

Absence of the ON keyword object name option implies all entities that the user attempts to access. A single logging statement may contain up to 20 objects.



BEGIN LOGGING Statement



Operation Any function for which an access right can be granted (e.g., GRANT).

BY *username* – implies all users, if not specified.

ON *object-name* – implies all entities, if not specified. Valid object-names are:

DATABASE	<i>database_name</i>	USER	<i>user_name</i>
TABLE	<i>table_name</i>	VIEW	<i>view_name</i>
MACRO	<i>macro_name</i>	PROCEDURE	<i>procedure_name</i>
TRIGGER	<i>trigger_name</i>	FUNCTION	<i>function_name</i>

END LOGGING Statement

Stops the auditing of SQL requests that attempt to access data that was started with a BEGIN LOGGING statement.

The END LOGGING statement erases only the frequency or text flags for the specified actions and user or object. However, if erasing a frequency leaves all logging blank for a particular user, database, and table, then the row is deleted from the AccLogRuleTbl table.

Use of the END LOGGING statement results in an error if BEGIN LOGGING is not currently in effect for the community for which logging is to be ended.

The END LOGGING statement has a number of options. Several are described below:

DENIALS — tracks only those entries when statement execution fails because the user does not have the privilege(s) necessary to execute the statement.

ALL — tells the system to make a log entry when the user attempts certain actions against the specified object, including:

CD	CREATE DATABASE	DP	DUMP	I	INSERT
CM	CREATE MACRO	D	DELETE	S	RETRIEVE/SELECT
CP	CHECKPOINT	CU	CREATE USER	RS	RESTORE
CT	CREATE TABLE	DU	DROP USER	U	UPDATE
CV	CREATE VIEW	G	GRANT	E	EXECUTE
DD	DROP DATABASE/USER				

BY username — lists the users for which to end logging on. The default is all users.

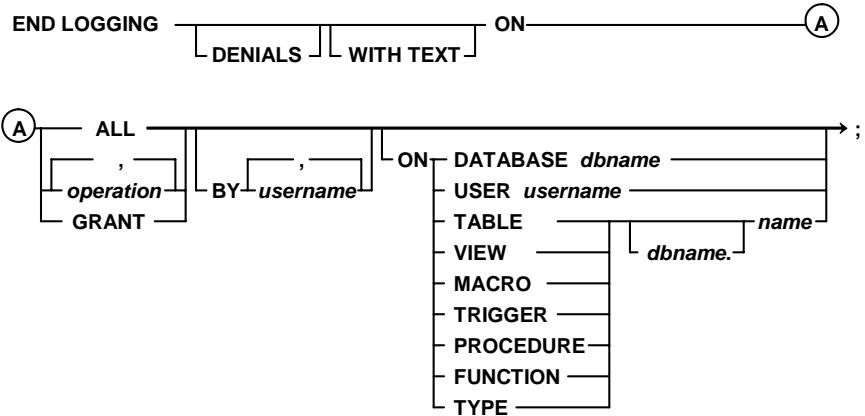
ON keyword object-name — defines which objects to end logging for in the log table when a user attempts to access them. The keyword object-name combinations must be one of the following:

USER	User name	TRIGGER	Trigger name
DATABASE	Database name	MACRO	Macro name
TABLE	Table name	PROCEDURE	Stored procedure name
VIEW	View name	FUNCTION	User-defined function name

Absence of the ON keyword object name option implies all entities that the user attempts to access. A single logging statement may contain up to 20 objects.



END LOGGING Statement



Operation Any function for which an access right can be granted (e.g., GRANT).

BY *username* – implies all users, if not specified.

ON *object-name* – implies all entities, if not specified. Valid object-names are:

DATABASE	<i>database_name</i>	USER	<i>user_name</i>
TABLE	<i>table_name</i>	VIEW	<i>view_name</i>
MACRO	<i>macro_name</i>	PROCEDURE	<i>procedure_name</i>
TRIGGER	<i>trigger_name</i>	FUNCTION	<i>function_name</i>

Setting up Access Logging

Before you can execute BEGIN/END LOGGING statements, you must run DIP script located on the software release medium. The DIP script creates a special security macro called DBC.AccLogRule. After you run the script, you must reset the system to initialize the logging software.

Log Entries

A logging entry does not indicate that the system successfully executed an SQL statement. It only indicates that the system checked the privileges necessary to execute the statement.

Keywords and Object-Names

By default, access logging inserts a row whenever a user accesses any database object. To restrict the scope of the log entries, you can include one of the following combinations in the BEGIN LOGGING statement:

DATABASE databasename	USER username
TABLE databasename.tablename	VIEW databasename.viewname
MACRO databasename.macroname	TRIGGER databasename.triggername
PROCEDURE databasename.procedurename	
FUNCTION databasename.functionname	TYPE databasename.datatypename

To activate access logging:

- Install Access Logging on the system with the DIP script DIPACC.
- Create and empower a security administrator.
- Submit the BEGIN LOGGING statement to define an access logging rule. You must submit this statement for each rule you define.

Example

Step 3 on the facing page is an example of three access logging rules. Each rule is described below:

- Log all attempts to access the security macros — Creates a new row in the log table each time a user attempts to access DBC.LogonRule or DBC.AccLogRule security macros. The ALL keyword indicates that any one of 27 user actions triggers an entry in the log table. The WITH TEXT option stores SQL statement contents in the log table.
- Log all denied attempts to access system user DBC — Logs all denied attempts made by any user to access system user with any one of the 27 defined actions. Teradata stores the SQL text in the log table.
- Log any SQL statements that involve CREATE or DROP USER/DATABASE or GRANT.



Setting Up Access Logging

STEP 1

Install Access Logging on the system:

1. Run the DIP script DIPACC to install the DBC.AccLogRule macro in system user DBC.
2. Restart the database to activate the code.

STEP 2

Create and empower a security administrator:

```
CREATE USER SecAdmin AS Password = secpasswd, PERM = 0, SPOOL = 200e6 ;
GRANT EXECUTE ON DBC.AccLogRule TO SecAdmin
GRANT EXECUTE ON DBC.LogonRule TO SecAdmin ;
```

STEP 3

Define access logging rules. Examples:

1. Log all attempts to access security macros.
2. Log all denied attempts to access DBC User.
3. Log any CREATE or DROP USER/DATABASE or GRANT commands.

```
BEGIN LOGGING WITH TEXT          ON EACH ALL
                                ON MACRO DBC.LogonRule,
                                MACRO DBC.AccLogRule;
```

```
BEGIN LOGGING DENIALS WITH TEXT ON EACH ALL
                                ON USER DBC;
```

```
BEGIN LOGGING WITH TEXT          ON EACH DATABASE, USER, GRANT;
```

Access Log Views

There are actually four system-supplied views that provide information about the entries in the DBC.AccLogRuleTbl[V] and the DBC.AccLogTbl[V]. They are:

DBC.AccLogRules[V]

Teradata maintains entries in this view's underlying table as the result of executing BEGIN/END LOGGING statements. The system uses these entries to determine which privilege checks should generate rows in the DBC.AccLogTbl.

DBC.AccessLog[V]

The underlying table for this view is DBC.AccLogTbl. Each entry in DBC.AccLogTbl indicates the results of a privilege check performed against a Teradata SQL request, based on the criteria defined by the BEGIN LOGGING statement.

Underlying DD/D Tables

- DBC.AccLogRuleTbl
- DBC.AccLogTbl



Access Log Views

DBC.AccLogRules[V]

Contains current logging rules generated by BEGIN and END LOGGING statements.

DBC.AccessLog[V]

Contains log entries collected as a result of applying access log rules.

Dictionary Tables Accessed:

- **DBC.AccLogRuleTbl**
- **DBC.AccLogTbl**

DBC.AccLogRules View

The DBC.AccLogRules[V] views provide information about logging rules currently in effect on the system. These rules were put into effect by successfully processing BEGIN LOGGING statements.

Example

The SQL statement on the facing page requests a list of the current rules stored in the DBC.AccLogRules table. It limits the rules to CREATE and DROP database and user, GRANT, SELECT, and EXECUTE.

The response produces four rows. Each contains a series of codes under each privilege column. There are three positions under each privilege. The first position indicates how often to log privilege checks. The second position indicates how often to log denials. The third position indicates when to save text. The following codes are used for positions 1 and 2:

B	Log FIRST and LAST occurrences.
E	Log each occurrence.
F	Log the FIRST occurrence.
L	Log the LAST occurrence.
Blank	No logging

The third position for text uses the following codes:

-	Save text only for Denial entries.
+	Save text for all entries.
=	Save text for all entries specified in multiple BEGIN LOGGING Statements

The code E+ means insert a row for each occurrence and save the text. The code E- means insert a row for each occurrence but only save the text for denials.



DBC.AccLogRules View

DBC.AccLogRules[V] – Returns information about current access logging rules.

UserName	DatabaseName	TVMName	AcrAlterFunction (AFN)
AcrCheckpoint (CPT)	AcrCreateDatabase (CDB)	AcrCreateFunction (CFN)	AcrCreateMacro (CMC)
AcrCreateTable (CTB)	AcrCreateUser (CUS)	AcrcreateView (CVW)	AcrCreateProcedure (CSP)
AcrCreExtProcedure (CXP)	AcrDelete (DEL)	AcrDropDatabase (DDB)	AcrDropFunction (DFN)
AcrDropMacro (DMC)	AcrDropTable (DTB)	AcrDropUser (DUS)	AcrDropView (DVW)
AcrDropProcedure (DSP)	AcrDump (DMP)	AcrExecute (EXE)	AcrExecuteFunction (EFN)
AcrExecuteProcedure (ESP)	AcrGrant (GRT)	AcrIndex (IDX)	AcrInsert (INS)
AcrReference (REF)	AcrRestore (RST)	AcrSelect (SEL)	AcrUpdate (UPD)
AcrCreateTrigger (CTG)	AcrDropTrigger (DTG)	AcrCreateRole (CRO)	AcrDropRole (DRO)
AcrCreateProfile (CPR)	AcrDropProfile (DPR)	AcrAlterProcedure (ASP)	AcrRepControl (REP)
AcrAlterExtProcedure (AXP)	AcrUDTUsage (USG)	AcrUDTType (UDT)	AcrUDTMethod (UDM)
AcrCreAuthorization (CAU)	AcrDropAuthorization (DAU)	CreatorName	CreateTimeStamp

ACR Columns are positional:

Position #1 = How often to log requests (F, L, B, E, blank = First, Last, Both, Each, None)

Position #2 = How often to log denials (F, L, B, E, blank = First, Last, Both, Each, None)

Position #3 = How often to save text (+ All entries, - Denials, = All Specified)

```
SELECT UserName      (CHAR (6)) AS "User//Name"
      ,DatabaseName (CHAR (6)) AS "Dbase//Name"
      ,TVMName       (CHAR (10)) AS "TVM//Name"
      ,AcrCreateDatabase, AcrCreateUser, AcrDropDatabase
      ,AcrDropUser, AcrGrant, AcrSelect, AcrExecute
FROM   DBC.AccLogRules;
```

User	Dbase	TVM	CDB	CUS	DDB	DUS	GRT	SEL	EXE
Name	Name	Name							
All	All	All	E +	E +	E +	E +	E +	E +	
All	DBC	LogonRule						E +	E +
All	DBC	All	E -	E -	E -	E -	E -	E -	E -
All	DBC	AccLogRule						E +	E +

Example Results:

BEGIN LOGGING – Example

The facing page provides an additional example of entries that may appear in the DBC.AccLogRules view. This view provides information about logging rules currently in effect on the system. These rules were put into effect by successfully processing BEGIN LOGGING statements.



BEGIN LOGGING – Example

```

BEGIN LOGGING DENIALS WITH TEXT ON EACH SELECT          ON TABLE PD.Employee; 1
BEGIN LOGGING           WITH TEXT ON FIRST INSERT      ON TABLE PD.Employee; 2
BEGIN LOGGING           WITH TEXT ON FIRST AND LAST DELETE ON TABLE PD.Employee; 3
BEGIN LOGGING           WITH TEXT ON FIRST UPDATE     ON TABLE PD.Employee; 4
BEGIN LOGGING DENIALS WITH TEXT ON LAST UPDATE        ON TABLE PD.Employee;

SELECT      UserName      (CHAR (6))      AS "User//Name"
            ,DatabaseName (CHAR (6))      AS "Dbase//Name"
            ,TVMName       (CHAR (10))     AS "TVM//Name"
            ,AcrSelect, AcrInsert, AcrDelete, AcrUpdate
FROM        DBC.AccLogRules
WHERE       DatabaseName = 'PD';
  
```

1 2 3 4

User Name	Dbase Name	TVM Name	SEL	INS	DEL	UPD
All	PD	Employee	E-	F +	B +	FL=

Position #1 = How often to log requests (F, L, B, E, blank = First, Last, Both, Each, None)

Position #2 = How often to log denials (F, L, B, E, blank = First, Last, Both, Each, None)

Position #3 = How often to save text (+ All entries, - Denials, = All Specified)

DBC.AccessLog View

The DBC.AccessLog[V] views display the entries made in the DBC.AccLogTbl system table. It returns information on the results of privilege checks performed against user requests to access data, which are logged as determined by the access logging rules.

Administrators may use this view to analyze application performance. This view would provide information about SQL requests (the text), tables and views accessed, embedded view (view of views), etc.

- Access Type
The same codes are used to indicate an access right, but with CUS and DUS for CREATE/DROP USER, AN for any privilege (validated for HELP and SHOW commands), HR for HOST UTILITY LOCK, and WL for WRITE LOCK.
- Frequency
F, L, B, E = First, Last, Both or Each.

Once logging begins, the access log grows very quickly. To keep space consumption under control, you should archive and empty the log regularly using the DBC.DeleteAccessLog view.

Examples:

DELETE FROM DBC.DeleteAccessLog;

- Deletes entries from DBC.AccLogTbl older than 30 days.

DELETE FROM DBC.DeleteAccessLog WHERE LOGDATE < (DATE - 90);

- Deletes entries from DBC.AccLogTbl older than 90 days.



DBC.AccessLog View

Displays entries made to DBC.AccLogTbl.

[DBC.AccessLog\[V\]](#)

LogDate	LogTime	LogonDate	LogonTime
LogicalHostID	IFPNo	SessionNo	UserName
AccountName	OwnerName	AccessType	Frequency
EventCount	AccLogResult	Result	DatabaseName
TVMName	ColumnName	StatementType	StatementText
QueryBand			

Access Type The same codes are used that indicate an access right.
Frequency F, L, B, E = First, Last, Both or Each.

To delete entries in the DBC.AccLogTbl, use the [DBC.DeleteAccessLog\[V\]\[X\]](#) views.

- **DELETE FROM DBC.DeleteAccessLog;**
 - Deletes entries older than 30 days.

- **DELETE FROM DBC.DeleteAccessLog WHERE LOGDATE < (CURRENT_DATE – 90);**
 - Deletes entries older than 90 days.

DBC.AccessLog View – Example

The SELECT statement on the facing page requests the contents of the DBC.AccLogTbl via the DBC.AccessLog view. The response shows seven separate entries.

User TFACT01 executed the following SQL statements at the listed times:

09:04:25 SELECT * FROM PD.Employee;

09:17:04 UPDATE PD.Employee SET Salary_Amount = 51000 WHERE Employee_Number = 100996;

09:24:31 UPDATE PD.Employee SET Salary_Amount = 51000 WHERE Employee_Number = 100995;

09:32:50 UPDATE PD.Employee SET Salary_Amount = 51000 WHERE Employee_Number = 100994;

Note that only the last UPDATE denial for TFACT01 appears on the following page. The rule specified to log the Last Update denial.

User Sysdba executed the following SQL statements at the listed times:

09:10:17 INSERT INTO PD.Employee VALUES
 (101001, 1060, 100991, 3054, 'Scott', 'Bill', 50000.00);

09:12:22 DELETE FROM PD.Employee
 WHERE Employee_Number = 100900;

09:12:25 DELETE FROM PD.Employee
 WHERE Employee_Number = 100901;

09:12:31 DELETE FROM PD.Employee
 WHERE Employee_Number = 100902;

09:15:54 UPDATE PD.Employee SET Salary_Amount = 51000
 WHERE Employee_Number = 101001;

09:57:20 UPDATE PD.Employee SET Salary_Amount = 51000
 WHERE Employee_Number = 100800;

Note that only the first and last Deletes for Sysdba appear on the following page based on the Access Log Rules. Also notice that only the first UPDATE appears on the facing page.

Note about uppercase and lowercase Frequency values:

When a FIRST and LAST are both specified for an entry, a lower case 'l' or 'f' are used to identify which entry this one is. An uppercase 'L' and 'F' is used if one is specified without the other.



DBC.AccessLog View – Example

Example:

List all of the entries
in the Access Log
table for the current
date.

```
SELECT      LogTime
           ,UserName (CHAR (10)) AS "User//Name"
           ,AccessType          AS "Access//Type"
           ,Frequency           AS "Log//Freq"
           ,AccLogResult        AS "Granted//Denied"
           ,StatementText       AS "Statement//Text"
FROM        DBC.AccessLog
WHERE       LogDate = CURRENT_DATE
ORDER BY   LogDate, LogTime ;
```

**Example
Results:**

LogTime	User Name	Access Type	Log Freq	Granted Denied	Statement Text
09:04:25	TFACT01	S	E	D	SELECT * FROM PD.Empl...
09:10:17	SYSDBA	I	F	G	INSERT INTO PD.Employee ...
09:12:22	SYSDBA	D	f	G	DEL FROM PD.Employee ...
09:12:31	SYSDBA	D	I	G	DEL FROM PD.Employee ...
09:15:54	SYSDBA	U	F	G	UPDATE PD.Employee SE ...
09:17:04	TFACT01	U	f	D	UPDATE PD.Employee SE ...
09:32:50	TFACT01	U	I	D	UPDATE PD.Employee SE ...

Note: The facing page contains the SQL that generated this report.

END LOGGING – Example

The facing page provides the END LOGGING statements to remove the logging for the PD.Employee table.

The BEGIN LOGGING statements (that were previously executed) are:

```
BEGIN LOGGING DENIALS WITH TEXT  
  ON EACH SELECT  
  ON TABLE PD.Employee;
```

```
BEGIN LOGGING WITH TEXT  
  ON FIRST INSERT  
  ON TABLE PD.Employee;
```

```
BEGIN LOGGING WITH TEXT  
  ON FIRST AND LAST DELETE  
  ON TABLE PD.Employee;
```

```
BEGIN LOGGING WITH TEXT  
  ON FIRST UPDATE  
  ON TABLE PD.Employee;
```

```
BEGIN LOGGING DENIALS WITH TEXT  
  ON LAST UPDATE  
  ON TABLE PD.Employee;
```



END LOGGING – Example

Previously, these rules were created for logging on PD.Employee table.

```
BEGIN LOGGING DENIALS WITH TEXT ON EACH SELECT      ON TABLE PD.Employee;
BEGIN LOGGING          WITH TEXT ON FIRST INSERT    ON TABLE PD.Employee;
BEGIN LOGGING          WITH TEXT ON FIRST AND LAST DELETE ON TABLE PD.Employee;
BEGIN LOGGING          WITH TEXT ON FIRST UPDATE    ON TABLE PD.Employee;
BEGIN LOGGING DENIALS WITH TEXT ON LAST UPDATE     ON TABLE PD.Employee;
```

To end the logging for PD.Employee table, the following statements can be executed:

```
END LOGGING DENIALS           ON SELECT, UPDATE      ON TABLE PD.Employee;
END LOGGING                  ON INSERT, DELETE, UPDATE ON TABLE PD.Employee;
```

To verify the rules have been removed, use the DBC.AccLogRules view:

```
SELECT      UserName        (CHAR (6))      AS "User//Name"
            ,DatabaseName   (CHAR (6))      AS "Dbase//Name"
            ,TVMName        (CHAR (10))     AS "TVM//Name"
            ,AcrSelect, AcrInsert, AcrDelete, AcrUpdate
FROM        DBC.AccLogRules
WHERE       DatabaseName = 'PD';
```

User	Dbase	TVM	SEL	INS	DEL	UPD
Name	Name	Name				

Rules for PD.Employee have been removed.

Teradata Administrator – Tools Menu > Access Logging

The Tools menu provides the following options.

Menu Selection	Function / Options
Create	Create an entirely new object – Database, Table, User, Profile, or Role.
Grant/Revoke	Grant or revoke general access privileges to users. Options include Object Rights, System Rights, Logon Rights, or Column Rights.
Administer Profiles	Create and manage Profiles for users. (V2R5 feature)
Administer Roles	Create and manage Roles. (V2R5 feature)
Clone User	Create a new user either identical or closely related to an existing user.
Modify User	Change the specifications of an existing user.
Access Logging	Create and manage Access Log rules.
Query Logging	Create and manage Query Log rules. (V2R5 feature)
Move Space	Reallocate permanent disk space from one database to another (efficient if not a direct descendent or parent).
Query	Create, modify, test, or run SQL query scripts.
Options	Configure the operational preferences for Teradata Administrator.

The example on the facing page effectively causes the following BEGIN LOGGING statement to be executed.

**BEGIN LOGGING WITH TEXT ON EACH
CREATE DATABASE, CREATE USER, CREATE PROFILE, CREATE ROLE, DROP
DATABASE, DROP USER, DROP PROFILE, DROP ROLE;**

TERADATA
Raising Intelligence

Teradata Administrator Tools Menu > Access Logging

Teradata Administrator can be used to Begin and End Access Logging – effectively managing Access Log rules.

To select ...
Tools >
Access Logging

Group Options:
Selecting different Groups will automatically choose specific Normal, Create, or Drop functions.

The corresponding BEGIN LOGGING statement is provided on the facing page.

Groups	Normal	Create	Drop
All	<input type="checkbox"/> Execute	<input type="checkbox"/> Index	<input type="checkbox"/> Table
Dictionary	<input type="checkbox"/> Select	<input type="checkbox"/> References	<input type="checkbox"/> View
Access	<input type="checkbox"/> Insert	<input type="checkbox"/> Dump	<input type="checkbox"/> Macro
Maintenance	<input type="checkbox"/> Update	<input type="checkbox"/> Restore	<input checked="" type="checkbox"/> Database
	<input type="checkbox"/> Delete	<input type="checkbox"/> Checkpoint	<input checked="" type="checkbox"/> User
	<input type="checkbox"/> Grant		<input type="checkbox"/> Trigger
	<input type="checkbox"/> Exec Procedure	<input type="checkbox"/> Alter Procedure	<input type="checkbox"/> Procedure
		<input type="checkbox"/> Profile	<input checked="" type="checkbox"/> Profile
		<input type="checkbox"/> Role	<input type="checkbox"/> Function
			<input type="checkbox"/> External Proc

Query Logging (DBQL) Concepts

Provides Collection of Historical records based on Rules

DBQL provides a series of predefined tables that can store, based on rules you specify, **historical records** of queries and their duration, performance, and target activity. DBQL is flexible enough to log information on the variety of SQL requests that run on Teradata, from short transactions to longer-running analysis and mining queries. **You begin and end collection for a user or group of users and/or one or a list of accounts.**

Performance

The performance of collecting query activity with DBQL is much better than attempting to capture query activity with the Access Logging facility because DBQL stores information in cache memory and will write to disk only when cache is full or when you use END QUERY LOGGING.

Notes about performance:

- The impact of turning on DBQL for all-AMP operations is negligible in comparison to the time an all-AMP operation takes to complete.
- The impact of turning on DBQL for single-AMP (PI) and two-AMP (USI) operations does impact the response time of the operation by a small amount. These types of operations are best suited for summary logging where the overhead is negligible.
- Specific information from the San Diego performance group follows:

With a single session of single-AMP queries (a total of 40,000 were executed), with default logging, the total response time increased by 6.5%. The performance report shows an average CPU path length increase of 4.6 ms for similar type work, from a baseline (no logging) path length of 4.3 ms, essentially a doubling of the path length.

While the response time tests indicated summary level logging on single AMP queries to have only a 1.6% increase in total response time, the performance report showed less than 1% increase in path length with summary logging.

The clear conclusion to be drawn is that single or few AMP queries will be best suited for summary logging, and will exhibit negligible overhead if logging is at that level.

With all-AMP queries, even very short ones, the response time tests were not able to pick up any overhead at all with any level of query logging enabled, including “all”.



Query Logging (DBQL) Concepts

- DBQL is a feature that is used to log historical query information.
 - DBQL caches and eventually stores query information in multiple Teradata Data Dictionary tables as the queries are executed.
 - Not intended for live review of queries and query steps (use Performance Monitor).
- Logging is invoked and revoked via SQL statements (**BEGIN QUERY LOGGING ...** and **END QUERY LOGGING ...**)
 - Logging can be invoked for all users, a list of users, a list of account strings or a particular user with one or more account strings.
- By default, 1 row per query is logged that contains user id information and some statistics for that query.
 - Options are available to expand the amount and kind of information to be logged.
- Notes and limitations include
 - DBQL views have numerous column changes in Teradata 12.0
 - DBSCtrl option (**DBQLFlushRate**) – determines the frequency for writing DBQL cache entries to DBQL dictionary tables. Default is 10 minutes.
 - Key use is to track SQL (note that TPump sends SQL to Teradata)
 - Teradata 12.0 feature – captures # of rows loaded by FastLoad and MultiLoad

Objects used in Defining Rules for DBQL

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.

Rules

You define rules that identify for which users and how much data to log for queries. For instance, you can log the first 5,000 characters of any query that runs during a session invoked by a specific user under a specific account. This rule can also be qualified so that only queries that exceed a specified time threshold are logged and those queries that execute in less than the threshold time are simply counted.

The DBC.DBQLRuleTbl table stores the rules resulting from each BEGIN QUERY LOGGING statement. One row exists for each set of specifications, which are made up of *user* and/or *account* plus any options or limits set for the user.

The DBC.DBQLRuleCountTbl table is an internal table that stores the cardinality of DBC.DBQLRuleTbl table.

The DBC.DBQLRules[V] views are used to display DBQL rules that are in effect.



Objects used in Defining Rules for DBQL

Users who are granted EXECUTE permission on the following macro can use the BEGIN QUERY LOGGING and END QUERY LOGGING statements.

Example:

```
GRANT EXECUTE ON DBC.DBQLAccessMacro TO  
Sysdba;
```

Initially, only DBC and SystemFE users are allowed to issue BEGIN/END QUERY LOGGING statements.

Execution of **BEGIN QUERY LOGGING** or **END QUERY LOGGING** statements causes rows (representing the rules) to be added or updated in ...

To view the rules in this table, **SELECT** from this view.

DD/D Macro

DBC.DBQLAccessMacro

DD/D Tables

DBC.DBQLRuleTbl

DD/D View

DBC.DBQLRules[V]

Note: There are additional tables and views that are used to hold and view captured query data – these are shown on next page.

Objects used in DBQL (cont.)

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.

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 can choose whether you want to populate the DBQL tables.

If you choose not to use the feature, the tables remain empty. If you want to use the feature, simply submit a BEGIN/END QUERY LOGGING statement, with or without options, to control the start, magnitude, and end of logging activity. The options enable you to control the volume and detail of the logged data. You can define rules, for instance, that log the first 5,000 characters of any query that runs during a session invoked by a specific user under a specific account, if the time to complete that query exceeds the specified time threshold.

The key or foundation table in DBQL is the DBC.DBQLogTbl table that holds the default rows. When you specify options that result in more information being captured, a default row is still generated in this table plus one or more additional logs (tables) will get rows.

- Exceptions to this are when you use the SUMMARY option or a query completes within a THRESHOLD. In these cases, default rows won't be placed into DBC.DBQLogTbl.

The facing page summarizes the tables and views used by DBQL to hold query data. Details on these views can be found in Appendix C.

Acronym: TDWM – Teradata Dynamic Workload Manager



Objects used in DBQL (cont.)

The views and associated tables used to hold query data are ...

<u>DD/D Views</u>	<u>DD/D Tables</u>	<u>DBQL Purpose</u>
DBC.QryLog[V]	DBC.DBQLLogTbl	Stores default rows (key table)
DBC.QryLogSteps[V]	DBC.DBQLStepTbl	One row per step
DBC.QryLogObjects[V]	DBC.DBQLObjTbl	One row per object referenced in query
DBC.QryLogSQL[V]	DBC.DBQLSqlTbl	Stores full SQL text – multiple rows may be needed
DBC.QryLogSummary[V]	DBC.DBQLSummaryTbl	Queries meeting Summary or Threshold rules
DBC.QryLogExplain[V]	DBC.DBQLExplainTbl	Stores EXPLAIN steps of query

Additional DBQL views that provide TDWM information include:

- DBC.QryLogEvents[V] – TDWM events that could affect system performance
- DBC.QryLogEventHis[V] – history of TDWM events; new with Teradata 12.0
- DBC.QryLogExceptions[V] – query exceptions as defined by Workload Definitions (WD)
- DBC.QryLogTDWM[V] – contains TDWM information for a query
- DBC.QryLogTDWMSum[V] – summary of WD activity for a predetermined period.

BEGIN QUERY LOGGING Statement

The DBQL facility is controlled by the Teradata SQL statements BEGIN QUERY LOGGING and END QUERY LOGGING.

There are numerous collection options using the WITH and LIMIT options. These options will be described on the following pages.

BEGIN QUERY LOGGING

When submitted by a user with EXECUTE privileges on DBC.DBQLAccessMacro, enables logging for the named users and/or accounts. For active sessions, logging begins when the next query is received. (NCR recommends a maximum of 100 user/account pairs per statement.)

When you do not specify a LIMIT option, one default row of query-level information is logged in DBQLogTbl for each query processed during a session that is initiated by any user for whom a query logging rule exists.

Default rows are stored in DBQLogTbl, the foundation of the DBQL feature. If you specify options that result in more detailed information, a default row is still generated in DBQLogTbl (except with the SUMMARY option or a query that completes within the limit specified with the THRESHOLD option), plus one or more additional logs are populated with one or more additional rows.

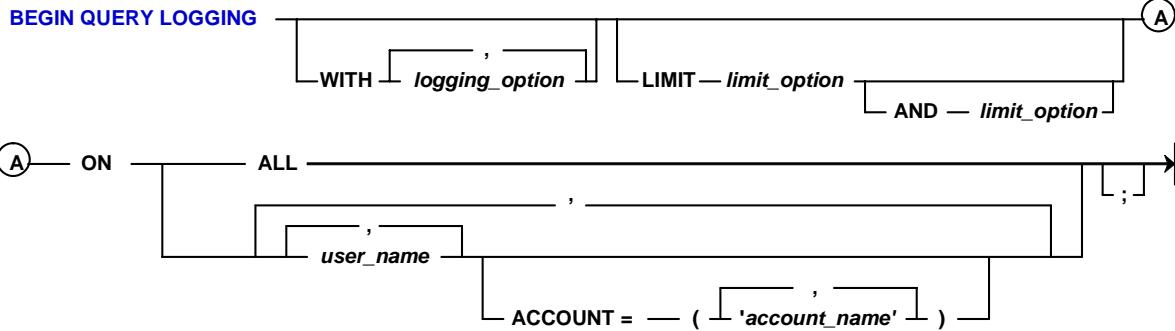
The Default Row

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 the following fields.

- 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.
- Time of receipt
- Number of processing steps completed
- Time the first step was dispatched, and
- Times of the first and last response packets were returned to the host.



BEGIN QUERY LOGGING Statement



- A BEGIN QUERY LOGGING statement **without** the WITH or LIMIT options causes default rows to be placed in the DBQLogTbl. **A default row contains:**
 - User name, account string (expanded), time stamp information
 - Unique ID for process, session, and client (host) connection
 - First 200 characters of SQL statement
- Use of the **WITH option(s)** cause a default row to be placed in DBC.DBQLogTbl **plus** additional rows in other DBQL tables.
- The **LIMIT option** may be used to limit the amount of SQL text captured, set thresholds, or just capture summary information.

BEGIN QUERY LOGGING WITH ... (cont.)

Note: Each of the following options logs the default row plus additional rows depending on the option specified.

... WITH ALL

Logs all information generated by all the WITH rules (EXPLAIN, OBJECTS, SQL, & STEPINFO). No other WITH rule is necessary, because ALL generates:

- One default row per query in DBC.DBQLLogTbl
- One or more rows per query for the EXPLAIN text in the DBC.DBQLExplainTbl
- One row per target object per query in DBC.DBQLObjTbl
- One row per step per query in DBC.DBQLStepTbl
- One or more rows per complete SQL statement in DBC.DBQLSqlTbl

Caution: Use this option sparingly and only for selected users, because it can consume excessive CPU resources and grow the logs very quickly (consumes DBC PERM space). Also set LIMIT SQLTEXT=0 if you specify the WITH ALL option, which also logs the entire SQL statement in the DBC.DBQLSqlTbl table.

... WITH EXPLAIN

Use this option selectively because the performance cost of generating EXPLAINS can be expensive. This option generates and logs the unformatted EXPLAIN text for each query. It does not generate EXPLAIN text for queries preceded by the EXPLAIN modifier.

- Logs one or more rows into DBC.DBQLExplainTbl

... WITH OBJECTS

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.

- One row per target object per query in DBC.DBQLObjTbl

... WITH SQL

This option logs the entire SQL statement in the DBC.DBQLSqlTbl table. Large statements can cause multiple rows to be written in order to log the full query text.

- Set LIMIT SQLTEXT=0 if you specify the WITH SQL option, which also logs SQL.

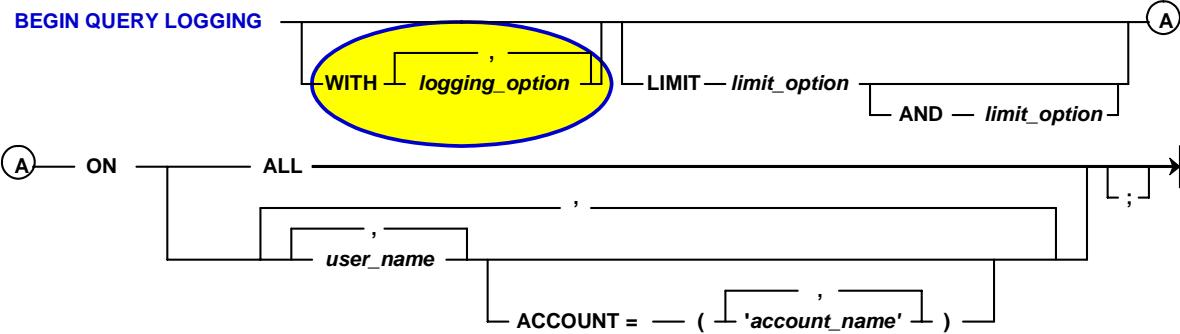
... WITH STEPINFO

Use this option selectively. Although step data is useful for analyzing queries, this option can generate many rows.

- This option inserts one row per step per query in the DBC.DBQLStepTbl.



BEGIN QUERY LOGGING WITH ... (Cont.)



- **WITH ALL**
 - includes all of the following options plus logs the default row.
- **WITH EXPLAIN (V2R6)**
 - logs the default row plus the unformatted EXPLAIN text for the query in the DBQLExplainTbl. Logging EXPLAIN text for all queries can have a performance impact.
- **WITH OBJECTS**
 - one row per target object per query in DBQLObjTbl plus default row in DBQLLogTbl.
- **WITH SQL**
 - logs the entire SQL for each request for each user being logged in DBQLSqlTbl plus default row in DBQLLogTbl.
- **WITH STEPINFO**
 - inserts one row per step per query in DBQLStepTbl plus default row.

BEGIN QUERY LOGGING LIMIT ... (cont.)

...LIMIT SQLTEXT

Use this option if you want to capture less than or more than the first 200 characters in the default row. To turn off text capture in the default row completely, specify 0 (zero). 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 SQL option, as many rows as needed to contain the full text will be logged in DBQLSqlTbl. (If you do this, define LIMIT SQLTEXT=0 to avoid redundant logging in both the default row and DBQLSqlTbl.)

Note: Also set LIMIT SQLTEXT=0 if you specify either the WITH ALL or the WITH SQL option, which also logs SQL.

...LIMIT SUMMARY

SUMMARY is useful for tracking voluminous short queries, such as for OLTP applications, because it does not grow the DBQLogTbl. It simply counts queries based on specified time differentials and stores the count results in DBQLSummaryTbl.

The SUMMARY option is unique in that it:

- Does not generate default rows in DBQLogTbl
- Summary information is flushed at system-controlled intervals of 10 minutes
- If no data has been collected for a summary category in a 10-minute interval, no rows will be written for it.

...LIMIT THRESHOLD

THRESHOLD also is useful for short, high-volume queries, but in addition to incrementing a count for qualifying queries, THRESHOLD logs a default row for any query that exceeds the specified time. This enables you examine the processing timestamps and the query structure. You can combine THRESHOLD with SQLTEXT if you want to capture more than the first 200 characters of a query that runs longer than THRESHOLD seconds for identification of the longer running queries.

You define the threshold of execution time, in seconds, which determines whether to log a query or just count it, as follows:

IF a query completes at or under the threshold time

- Increments the query count and the query seconds
- Stores the final count for the session as a row in DBQLSummaryTbl
- 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)

IF a query runs beyond the threshold time

- DBQL logs a default row for the query in DBQLogTbl so you can examine its structure and the number and level of processing steps.

TERADATA
Raising Intelligence

BEGIN QUERY LOGGING LIMIT ... (cont.)

The diagram illustrates the structure of the `BEGIN QUERY LOGGING` statement. It starts with `BEGIN QUERY LOGGING` followed by optional `WITH logging_option`, `LIMIT limit_option` (enclosed in a yellow oval), and `AND limit_option`. The next part is `ON ALL`, followed by a list of `user_name`s separated by commas, enclosed in a box. Finally, there is an optional `ACCOUNT = ('account_name')`.

- **LIMIT SQLTEXT** – specify the amount of SQL text to capture in the default row of DBQLogTbl. (Default is 200 char., 0 = off, max = 10,000 characters)
 - **LIMIT SUMMARY** (For short high volume queries – example tactical queries)
 - Counts queries; count is written in DBQLSummaryTbl every 10 min (if count > 0)
 - SUMMARY doesn't generate default rows in DBQLogTbl.
 - **LIMIT THRESHOLD** (Also for short high-volume queries – example tactical queries)
 - Similar to SUMMARY, but default rows are generated in DBQLogTbl.
 - Threshold, in seconds, determines whether to log a query or just count it.
 - Query that complete \leq threshold (sec.) are counted in DBQLSummaryTbl.
 - Query that complete $>$ threshold (sec.), DBQL logs the default row.

BEGIN QUERY LOGGING Examples

The facing page contains a number of BEGIN QUERY LOGGING examples.

When you specify **BEGIN QUERY LOGGING ON ALL**; you effectively create a rule for “everyone”. Therefore, you cannot create rules for specific users.

The opposite is also true. If you create rules for specific users, you cannot create a rule for ALL.

If you use the **BEGIN QUERY LOGGING ON ALL ACCOUNT = (...)**; then you can also create individual rules for specific users as long as you don't specify account IDs at the user level that are in the ALL account ID list.

These statements will complete successfully.

```
BEGIN QUERY LOGGING ON ALL ACCOUNT = ('$M');
```

```
BEGIN QUERY LOGGING ON tfact05;
```

```
BEGIN QUERY LOGGING ON tfact06 = ('$H');
```

The following will fail because ALL has been specified with '\$M'.

```
BEGIN QUERY LOGGING ON tfact07 = ('$M');
```



BEGIN QUERY LOGGING Examples

```
BEGIN QUERY LOGGING ON ALL;
```

- This creates a rule for all users – you will not be able to create rules for specific users.
- If rules exist for specific users/accounts, then you cannot create a rule for ALL.

```
BEGIN QUERY LOGGING ON tfact01, tfact02;
```

- This creates 2 rules – one for each specified user.
- You can END QUERY LOGGING for either or both of the users.

```
BEGIN QUERY LOGGING ON tfact03 ACCOUNT = ('$L_&D&H', '$M_&D&H');
```

- This creates 2 rules for a specific user – each rule has a specific account ID.

```
BEGIN QUERY LOGGING ON ALL ACCOUNT = ('$H', '$R');
```

- This creates 2 rules for all users – each rule identifies a specific account ID.
- You can END QUERY LOGGING for either or both of the account IDs.

In these examples, the WITH and LIMIT options aren't used.
Therefore, default rows will be created in the DBQLogTbl.

BEGIN QUERY LOGGING Examples (cont.)

The facing page contains additional BEGIN QUERY LOGGING examples.

Limits that may be used include:

SQLTEXT: option to control the number of SQL statement characters to log

- 200 characters of SQL logged in default row
- SQLTEXT values range from 0 to 10,000
- “SQLTEXT” without a value logs 10,000 characters

THRESHOLD: option to limit the queries logged by elapsed time

- THRESHOLD values are specified in seconds.
- “THRESHOLD” without a value results in a 5 second value.
- Queries greater than Threshold value generate a default row.
- Maximum THRESHOLD value is 32,767 seconds.

SUMMARY: option to only count running queries based on elapsed time.

- 3 values (in seconds) are needed
- No verification done on order of values
- 4 count intervals are created
 - $\leq n1$
 - $> n1 \leq n2$
 - $> n2 \leq n3$
 - $> n3$
- Data is logged in the DBC.DBQLSummaryTbl every 10 minutes (or if the cache should get full). There is no parameter to change the flush period of 10 minutes.
- 1 row for each count > 0
- Data is logged by session id
- **SUMMARY cannot be used with any other “limits”.**



BEGIN QUERY LOGGING Examples (cont.)

BEGIN QUERY LOGGING WITH OBJECTS, SQL ON ALL;

- Default rows are logged as well as complete SQL text and objects used in the query.
- Even though the complete SQL text is captured in DBQLSqlTbl, the default row still has the first 200 bytes of SQL text. This may help in recognizing the query in the QryLog.

BEGIN QUERY LOGGING WITH ALL ON ALL;

- ALL options are logged for ALL users (probably generates too much information).

BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 60, 600 ON ALL;

- Default Summary option is to only count running queries based on elapsed time. Counts are logged in DBQLSummaryTbl every 10 minutes or when cache is full.
- 3 values (in sec.) are required. 4 count intervals are logged (<=5, <=60, <=600, >600)
- Summary limit cannot be used with any other limits.

BEGIN QUERY LOGGING LIMIT THRESHOLD = 60 ON ALL;

- If a query runs for less than or equal to 60 seconds (1 minute), increment the count.
- If a query runs longer than 1 minute, log a default row.

In these examples, the ON option can also specify user name and/or account IDs.

BEGIN QUERY LOGGING Examples (cont.)

With V2R6, the SUMMARY and THRESHOLD limits have 2 new options. The default for each of these limits is elapsed time (if CPUTIME or IOCOUNT is not specified).

SUMMARY

SUMMARY=n1,n2,n3 [CPUTIME or IOCOUNT]

The SUMMARY option is designed for use with short, OLTP-like, queries. This option counts the number of queries for the session that fall into each of four time intervals. Interval values can be specified in seconds, CPU time, or I/O counts.

If you specify SUMMARY, then you cannot specify any other options. You must specify the first three intervals explicitly. The fourth interval is created by default.

CPUTIME – use the CPU time for the query to set ranges and to summarize. The SUMMARY value is in units of 0.01 second. For example, if you specify 500 for one of the intervals, then the value used to make the determination is 5 CPU seconds.

IOCOUNT – use the I/O count for the query to set ranges and to summarize.

THRESHOLD

THRESHOLD [=n] [CPUTIME or IOCOUNT]

This option is also designed for use with tactical queries. This option sets a threshold time in seconds that determines whether a query is to be logged fully or just counted.

If a query completes earlier than or equal to the threshold value, then it is only logged as a count in DBQLSummaryTbl. The Threshold row in DBQLSummaryTbl is identified by a HighHist field value of 0. If a query completes later than the threshold value, then a full entry is logged for it in DBQLogTbl with values for all fields of the row.

If you specify THRESHOLD without also specifying a value for n, then the value 5 seconds is assigned by default.

CPUTIME – use the CPU time for the query to determine whether to log or just count it. If you do not specify a threshold value for n, then the system uses the default CPUTIME value of 0.05 CPU seconds. The value is in units of 0.01 second. For example, if you specify 500, then the value used to make the determination is 5 CPU seconds.

IOCOUNT – use the I/O count for the query to determine whether to log or just count it. If you do not specify a threshold value for n, then the system uses the default IOCOUNT of 5.

The maximum value for n is 32767 seconds (roughly 9 hours).



BEGIN QUERY LOGGING Examples (cont.)

The default times specified with **THRESHOLD** and **SUMMARY** Limits represent “elapsed wall-clock time”.

With V2R6, the THRESHOLD AND SUMMARY Limits have two additional options:

- **CPUTIME** – uses actual CPU time to set ranges and to summarize – represented in units of 0.01 sec.
 - **IOCOUNT** – uses the I/O count for the query to set ranges and to summarize.

BEGIN QUERY LOGGING LIMIT SUMMARY = 100, 500, 2000 CPUTIME ON ALL;

- This Summary example only counts queries based on actual CPU time.
 - 100 is 1 CPU second, 500 is 5 CPU seconds, and 2000 is 20 CPU seconds. 3 values are required. 4 count intervals are logged (≤ 100 , ≤ 500 , ≤ 2000 , > 2000)
 - Summary limit cannot be used with any other limits.

BEGIN QUERY LOGGING LIMIT THRESHOLD = 50 IOCOUNT ON ALL;

- If a query generates less than or equal to 50 IO's, increment the count.
 - If a query generates more than 50 IO's, log a default row.

In these examples, the ON option can also specify user name and/or account IDs.

END QUERY LOGGING Statement

... *ON ALL* – to stop logging query information for all users specified by a rule created by a BEGIN QUERY LOGGING ON ALL statement.

...*user_name* – the name of a specific user or set of users for whom logging of SQL query information is to be stopped.

...*account_name* – the name of one or more specific accounts for which logging of SQL query information is to be stopped.

Account names must be enclosed by LEFT and RIGHT PARENTHESIS characters.

When you specify a list of accounts, each account name must be delimited by APOSTROPHE characters and separated by COMMA characters.

If you begin query logging on a specific user-account pair, then you must also specify that user-account pair to end query logging.

When this statement is submitted by a qualified user (with EXECUTE privileges on DBQLAccessMacro), logging is stopped for the named users and/or accounts. This command can be used for up to 100 active sessions. When this command is used, a routine is called that commits the data and flushes the cache.

When you enable or disable query logging, the change has an immediate effect on active sessions where the user or account being logged appears within the first 100 names you specify in the user and/or account list of a single BEGIN/END QUERY LOGGING statement. For users listed beyond the first 100, the user must log off from the Teradata Database and restart the session.

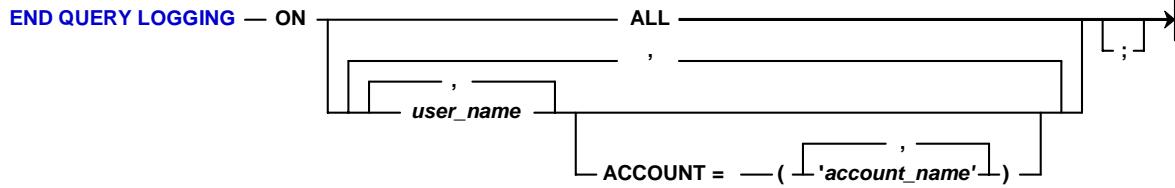
Note: If you need to enable or disable large volumes of users or accounts, the best practice is to submit an additional BEGIN/END QUERY LOGGING statement for each block of 100 names.

When you disable logging (submit an END QUERY LOGGING statement) for an active session (a query for that session is in process) and data is already cached, the following occurs:

- The data is committed immediately
- One or more DBQL rows are written (but may be incomplete)
- The cache is flushed
- Subsequent queries during that session are not logged



END QUERY LOGGING Statement



- If “ON ALL” was used in the BEGIN statement, “ON ALL” must be used in the END statement.
- If a list of users or a list of account strings was given in the BEGIN statement, logging can be ended on an individual basis.
- The “END QUERY LOGGING” statement will cause DBQL cache to be written to the tables except for Summary cache.

<pre>END QUERY LOGGING ON ALL; END QUERY LOGGING ON tfact01; END QUERY LOGGING ON tfact03 ACCOUNT=('\$L_&D&H');</pre>	(You can end logging for a specific user.) (You can end logging for a specific account of a user.)
---	---

DBC.DBQLRules View

A description of the columns in this view follows:

UserName:	Identifies the name associated with the rule.
AccountString:	Contains the default account for the user, or the specified Account.
ExplainFlag:	T = Explain text will be stored; F = No explain text is provided.
ObjFlag:	T = Object data (Columns, Indexes) will be stored; F = No object data.
SqlFlag:	T = SQL text will be stored; F = No SQL text is provided.
StepFlag:	T = Step level data will be stored; F = No step level data is provided.
SummaryFlag:	T = Summary information will be stored; F = not summarized.
ThresholdFlag:	T = Count queries shorter than ThreshValue seconds; detailed data on long queries. F = Detailed data for all queries
TextSizeLimit:	Indicates the number of characters of SQL text to be stored.
SummaryVal1:	If Summary or Threshold is T, low history in seconds, CPU or IO.
SummaryVal2:	Group 2: SummaryVal1 to SummaryVal2 seconds.
SummaryVal3:	Group 3: SummaryVal2 to SummaryVal3 seconds.
ThreshValue:	Indicates if values are elapsed time, CPU seconds, or IO counts 0 = Summary Values are in seconds, 1 = Values in CPU hundredths of seconds 2 = Values in IO count

The BEGIN QUERY LOGGING statements that generated the rules on the facing page are as follows:

```
BEGIN QUERY LOGGING ON ALL ACCOUNT = ('$H', '$R');
BEGIN QUERY LOGGING LIMIT SQLTEXT=500 ON tfact01, tfact02;
BEGIN QUERY LOGGING ON tfact03 ACCOUNT = ('$L_&D&H', '$M_&D&H');
BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;
BEGIN QUERY LOGGING LIMIT THRESHOLD = 200 CPUTIME ON tfact05;
BEGIN QUERY LOGGING WITH SQL LIMIT SQLTEXT=0 ON tfact06;
```

The END QUERY LOGGING statements that correspond to these BEGIN QUERY LOGGING statements are as follows:

```
END QUERY LOGGING ON ALL ACCOUNT = ('$H', '$R');
END QUERY LOGGING ON tfact01, tfact02;
END QUERY LOGGING ON tfact03 ACCOUNT = ('$L_&D&H', '$M_&D&H');
END QUERY LOGGING ON tfact04;
END QUERY LOGGING ON tfact05;
END QUERY LOGGING ON tfact06;
```

Note: Simply replacing the BEGIN with END will also remove the rules.

```
END QUERY LOGGING ON ALL ACCOUNT = ('$H', '$R');
END QUERY LOGGING LIMIT SQLTEXT=500 ON tfact01, tfact02;
END QUERY LOGGING ON tfact03 ACCOUNT = ('$L_&D&H', '$M_&D&H');
END QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;
END QUERY LOGGING LIMIT THRESHOLD = 200 CPUTIME ON tfact05;
END QUERY LOGGING WITH SQL LIMIT SQLTEXT=0 ON tfact06;
```



DBC.DBQLRules View

DBC.DBQLRules[V] – returns information about current query logging rules.

UserName	AccountString	ExplainFlag	ObjFlag	SqIFlag
StepFlag	SummaryFlag	ThresholdFlag	TextSizeLimit	SummaryVal1
SummaryVal2	SummaryVal3	ThreshValue		

Example:

```
SELECT    UserName      (CHAR (8))  AS "User"
          ,AccountString (CHAR (8))  AS "Acct_ID"
          ,SqIFlag        AS "Sql"   , TextSizeLimit   AS "Size"
          ,ThresholdFlag AS "T_Flag" , SummaryFlag    AS "S_Flag"
          ,ThreshValue    AS "Type"
          ,SummaryVal1   AS "V1"   , SummaryVal2   AS "V2" , SummaryVal3 AS "V3"
FROM      DBC.DBQLRules
ORDER BY 1;
```

Notes:

BEGIN QUERY
LOGGING
statements are on
facing page.

ThreshValue (Type):
0 – Elapsed Time
1 – CPUTIME
2 – IOCOUNT

User	Acct_ID	SqI	Size	T_Flag	S_Flag	Type	V1	V2	V3
All	\$H	F	200	F	F	?	?	?	?
All	\$R	F	200	F	F	?	?	?	?
tfact01		F	500	F	F	?	?	?	?
tfact02		F	500	F	F	?	?	?	?
tfact03	\$L_&D&H	F	200	F	F	?	?	?	?
tfact03	\$M_&D&H	F	200	F	F	?	?	?	?
tfact04		F	200	F	T	0	5	30	60
tfact05		F	200	T	F	1	200	?	?
tfact06		T	0	F	F	?	?	?	?

DBC.QryLog View – Example

Columns that are available in the DBC.QryLog view include:

ProcID	CollectTimeStamp
QueryID	UserID
UserName **	DefaultDatabase **
AcctString	ExpandAccString
SessionID	LogicalHostID
RequestNum	LogonDateTime
AccStringTime	AccStringHour
AccStringDate	AppID
ClientID	ClientAddress **
QueryBrand ***	ProfileID
StartTime	FirstStepTime
LastRespTime	ElapsedTime **
NumSteps	NumStepswPar
MaxStepsInPar	NumResultRows
TotalIOcount	TotalCPUTime
ErrorCode	ErrorText
WarningOnly **	AbortFlag
CacheFlag	QueryText
NumOfActiveAMPs **	HotAmp1CPU
HotCPUAmpNumber **	LowAmp1CPU
HotAmp1IO **	HotIOAmpNumber **
LowAmp1IO **	SpoolUsage

** New with V2R6

*** New with Teradata 12.0



DBC.QryLog View – Example

DBC.QryLog[V] – returns information about default rows in the DBQLogTbl.

Example of the data within one default row with a QueryID of 17866.

```
SELECT      ProcID, UserName, CollectTimeStamp, QueryID, UserID, AcctString,
            ExpandAcctString, LogonDateTime, StartTime, FirstStepTime, LastRespTime,
            ElapsedTime, NumSteps, QueryText
FROM        DBC.QryLog
WHERE       QueryID = 17866;
```

New with V2R6

Shown in BTEQ with SIDETITLES and FOLDLINE on.

Result:

```
ProcID 16383
UserName TFACT03
CollectTimeStamp 2006-01-19 19:39:56.96
QueryID 17866
UserID 0000B308
AcctString $M_9038_&S_&D&H
ExpandAcctString $M_9038_000124751_06011920
LogonDateTime 2006-01-19 19:41:09.82
StartTime 2006-01-19 20:10:48.56
FirstStepTime 2006-01-19 20:10:48.77
LastRespTime 2006-01-19 20:10:52.06
ElapsedTime 0:00:03.500000
NumSteps 5
QueryText SELECT * FROM DBC.AccessLog WHERE "Result" = 'G' and ...
```

DBC.QryLogSummary View – Example

Columns that are available in the DBC.QryLogSummary view are:

ProcID
CollectTimeStamp
UserID
AcctString **
LogicalHostID **
AppID **
ClientID **
ClientAddr **
QueryBand **
ProfileID
SessionID
QueryCount
ValueType **
QuerySeconds **
AverageTime **
TotalIOPCount **
AverageIO **
TotalCPUTime **
AverageCPU **
LowHist
HighHist

** New with V2R6

An example of the output from this view is provided on the facing page. This summary data was collected for the tfact04 user who had established multiple sessions on the system. The summary rule was created as follows:

BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;



DBC.QryLogSummary View – Example

Returns information about summary rows in the DBQLSummaryTbl.

This example is based on the summary rule:

BEGIN QUERY LOGGING LIMIT SUMMARY = 5, 30, 60 ON tfact04;

```
SELECT UserID, CollectTimeStamp, QueryCount, QuerySeconds,
       LowHist, HighHist
  FROM DBC.QryLogSummary
 ORDER BY 2, 3, 6;
```

For #2, the average time of queries can be calculated:

$$\begin{aligned} 34 / 113 &= .30 \\ 105 / 8 &= 13.13 \\ 121 / 3 &= 40.33 \\ 338 / 2 &= 169.00 \end{aligned}$$

Result:

UserID	CollectTimeStamp	QueryCount	QuerySeconds	LowHist	HighHist
00001504	2006-01-17 18:34:08	8	18	0	5
00001504	2006-01-17 18:34:08	1	17	5	30
00001504	2006-01-17 18:44:08	113	34	0	5
00001504	2006-01-17 18:44:08	8	105	5	30
00001504	2006-01-17 18:44:08	3	121	30	60
00001504	2006-01-17 18:44:08	2	338	60	32767
00001504	2006-01-17 18:54:08	5	3	0	5
00001504	2006-01-17 18:54:08	6	51	5	30
00001504	2006-01-17 18:54:08	1	141	60	32767

1 – In this summary collection, no queries were executed that exceeded 30 seconds.

2 – In this summary collection, queries were executed in all 4 summary intervals.

3 – In this summary collection, no queries were executed that ran between 30 and 60 seconds.

Teradata Administrator – Tools Menu > Query Logging

The Tools menu provides the following options.

Menu Selection	Function / Options
Create	Create an entirely new object – Database, Table, User, Profile, or Role.
Grant/Revoke	Grant or revoke general access privileges to users. Options include Object Rights, System Rights, Logon Rights, or Column Rights.
Administer Profiles	Create and manage Profiles for users. (V2R5 feature)
Administer Roles	Create and manage Roles. (V2R5 feature)
Clone User	Create a new user either identical or closely related to an existing user.
Modify User	Change the specifications of an existing user.
Access Logging	Create and manage Access Log rules.
Query Logging	Create and manager Query Log rules.
Move Space	Reallocate permanent disk space from one database to another (efficient if not a direct descendent or parent).
Query	Create, modify, test, or run SQL query scripts.
Options	Configure the operational preferences for Teradata Administrator.

The example on the facing page effectively causes the following BEGIN QUERY LOGGING statement to be executed.

```
BEGIN QUERY LOGGING LIMIT THRESHOLD=60
ON tfact01, tfact02, tfact03, tfact04, tfact05, tfact06;
```

TERADATA
Raising Intelligence

Teradata Administrator Tools Menu > Query Logging

Teradata Administrator can be used to Begin and End Query Logging – effectively managing DBQL rules.

To select ...

Tools > Query Logging

Options:

DISPLAY will show query log rules for the selected users.

The corresponding BEGIN QUERY LOGGING statement is provided on the facing page.

The screenshot shows the Teradata Administrator application window titled 'Teradata Administrator - [WXP_TD.DBC]'. On the left, there's a tree view under 'DBC' containing various database objects like 'All', 'console', 'Crashdumps', 'DBCMANAGER', 'Default', 'PUBLIC', 'QCD', 'Spool_Reserve', 'SysAdmin', 'Sysdba', 'SYSLIB', 'SYSUDTLIB', 'Sys_Calendar', 'TDPUSER', and 'tdwm'. A 'Ready' status message is at the bottom left. On the right, a modal dialog box is open titled 'Query Logging Rules [WXP_TD]'. It has a list of 'User(s)' including 'SysAdmin', 'Sysdba', 'SystemFe', 'Sys_Calendar', 'TDPUSER', 'tdwm', and several 'Itact...' entries. Below this is an 'Account(s)' section which is currently empty. Under 'Limits', there are checkboxes for 'SQL Length' (set to 200), 'Summary' (unchecked), and 'Threshold' (checked, set to 60). There are also radio buttons for 'Elapsed', 'Cpu time', and 'I/O count'. On the right side of the dialog, there's a group of 'Optional Information' checkboxes: 'Object Names', 'Full SQL Text', 'Step Information', 'Explain Text', and 'Everything'. At the bottom right of the dialog is a timestamp 'Elapsed 00:00:00'.

Access and Query Logging Summary

The facing page summarizes some important concepts regarding this module.



Access and Query Logging Summary

There are two logging facilities available to the database and/or security administrator.

- **Access Logging Facility**
 - Used for access and security audit analysis
- **Query Logging Facility (DBQL)**
 - Used for query activity and workload analysis
- **Teradata V2R6.2 DBQL Enhancements:**
 - Macros, views, triggers, stored procedures, and User-DefinedFunctions (UDFs) are now logged in DBQLObjTbl if the WITH OBJECTS option is used.
 - By querying DBQLObjTbl information, Database Administrators (DBAs) are able to see which views and macros users access.
 - If the WITH OBJECTS option is used, FastLoad and MultiLoad target tables are also logged in the DBQLObjTbl.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. In order to use the BEGIN/END LOGGING commands, what is the name of the system macro you need execute permission on?

2. How is this macro initially created?

3. What is a negative impact of the following statement?

BEGIN LOGGING WITH TEXT ON EACH

4. With DBQL, what is the size of the default text captured for queries? _____

5. True or False. With DBQL, the LIMIT SUMMARY option cannot be used with any other LIMIT.

6. True or False. With DBQL, use WITH SQL option only captures a maximum of 10,000 characters.

7. True or False. With DBQL, the option WITH ALL ON ALL is typically a good choice.

8. True or False. With DBQL, default rows are logged in the DBC.DBQLogTbl.

Lab Exercise 49-1

The following page contains the start of this lab exercise.



Lab Exercises

Lab Exercise 49-1

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to view information in the data dictionary about security defaults and invalid logons in the system (use Appendix C). This lab exercise also covers information discussed in Module 48.

Tasks

1. What system security defaults are in effect for your system?

Number of days to expire password:

Minimum number of characters required:

Maximum number of characters required:

Are digits allowed?

Yes _____ No _____

Are special characters allowed?

Yes _____ No _____

Maximum failed logons permitted (0=never lock):

Minutes to elapse before unlocking:

Days to expire before password reuse:

2. Are these the security defaults that are in effect for your username? Yes or No.
3. Is a Profile in effect for your username? If so, what is the name of your Profile? _____
4. If a Profile is being used, which attributes in the Profile override the system security defaults?

Lab Exercise 49-1 (cont.)

The following page continues this lab exercise.



Lab Exercises

Lab Exercise 49-1 (cont.)

5. Using the DBC.LogOnOff view, list the “BAD” logon attempts on your system that have occurred during the last ten days. Qualify the SELECT using LIKE 'BAD%'.

Number of Bad Logons (System) _____

Number of Bad Logons (Your UserName) _____

6. Display the logon rules, if any, currently in force on your system.

Lab Exercise 49-2

The following page contains the start of this lab exercise.



Lab Exercises

Lab Exercise 49-2

Purpose

In this lab, you will use BTEQ or (Teradata SQL Assistant) to list access and query logging rules that are being used and the entries in these logs for your username.

Tasks

1. Using the DBC.AccLogRules view, list the access log rules that are in effect for your username.

Which codes are being logged and what type of logging is being captured?

<u>Code</u>	<u>Type of Logging</u>	<u>SQL Function Being Logged</u>
Ex. <u>CDB</u>	<u>B +</u>	<u>Create Database</u> _____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

2. How many different access logging rules are there for all users? Count _____

Lab Exercise 49-2 (cont.)

The following page continues this lab exercise.



Lab Exercises

Lab Exercise 49-2 (cont.)

Tasks

3. Execute the following statement.

```
CREATE DATABASE Test FROM DBC AS PERM=0;
```

(this command should fail – access right violation)

4. Using the DBC.AccessLog view, list the access log entries for the last two weeks for your username.

How many entries are in this log have been granted? _____

Note: The AccLogResult column (12.0) is named "Result" in previous releases.

How many entries are in this log have been denied? _____

What is the difference between the Create Table and the Execute command log entries?

Lab Exercise 49-2 (cont.)

The following page continues this lab exercise.



Lab Exercises

Lab Exercise 49-2 (cont.)

Tasks

5. Using the DBC.DBQLRules view, list the attributes of query log rule that is in effect for your username.

Explain Text Logged _____

Objects Logged _____

Full SQL Logged _____

Execution Steps Logged _____

Summary _____

If Summary, times _____ _____ _____

Threshold _____

If threshold, time _____

SQL Text Size _____

Threshold/Summary Type _____

6. Using the DBC.Qrylog view, list the logged queries for your username and determine how many queries are logged for your username.

Count = _____

Using this view, how many queries are logged for all of the users with usernames like 'TLJC%'?

Count = _____

Teradata Training

Notes

Module 50



Workload Management

After completing this module, you will be able to:

- Describe the type of TDWM rule to apply to limit certain kinds of queries.
- List the query attributes that are used to classify a query into a Workload Definition.
- Specify the exception actions that are possible for a workload definition.
- Place the TASM control options in the proper sequence as they are acted upon by Teradata software.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Levels of Workload Management.....	50-4
What is TASM?	50-6
Teradata Dynamic Workload Manager (TDWM).....	50-8
TDWM Query Management Architecture	50-10
Query Management Architecture (cont.)	50-12
TDWM Main Window – Types of Rules.....	50-14
Filters and Throttles for Query Management.....	50-16
Object Access and Query Resource Filters.....	50-18
Object and Load Utility Throttles	50-20
TDWM – General Settings	50-22
Object Access Filter Properties.....	50-24
Query Resource Filter Properties.....	50-26
Object Throttling Properties.....	50-28
Linking Objects to Rules.....	50-30
Workload Definitions.....	50-32
Creating Workloads	50-34
Example of Using Workloads	50-36
TDWM – Create Workload Definitions.....	50-38
WD – Classification Criteria.....	50-40
WD – Exception Criteria.....	50-42
TDWM – Specify Exception Criteria	50-44
Example – Exception Handling	50-46
TDWM – Priority Scheduler.....	50-50
Teradata Workload Analyzer	50-52
Teradata Query Scheduler.....	50-54
Summary	50-56
Review Questions	50-58

Levels of Workload Management

The facing page illustrates three tiers of workload management. This module provides details on the Teradata Dynamic Workload Manager (DWM or TDWM) and Teradata Query Scheduler (QS or TQS) utilities.

Teradata Dynamic Workload Manager (TDWM)

Teradata Dynamic Workload Manager (also known as Teradata DWM or TDWM) provides a graphical user interface (GUI) for creating rules that manage database access, increase database efficiency, and enhance workload capacity. Via the rules created through Teradata DWM, queries can be rejected, throttled, or executed when they are submitted to the Teradata Database.

Teradata Query Scheduler (QS) is designed to provide a facility to submit Teradata SQL jobs to the Teradata Database. TQS is not shown on the facing page, but is an external tool that simply submits SQL to Teradata.

Priority Scheduler

The Priority Scheduler is a resource management tool that controls how compute resources (e.g., CPU) are allocated to different users in a Teradata Database system. This resource management function is based on scheduler parameters that satisfy your site-specific requirements and system parameters that depict the current activity level of the Teradata Database system. You can provide Priority Scheduler parameters to directly define a strategy for controlling compute resources.

Database Query Log

The Database Query Log (DBQL) is a feature that lets you 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.

What Happened to TDQM?

Teradata Dynamic Query Manager (DQM) was the toolset used with Teradata V2R5.0 and V2R5.1 to perform query management and scheduled request functions. With Teradata V2R6.0, the features of Teradata DQM were redesigned and incorporated into these products: Teradata Dynamic Workload Manager, Teradata Query Scheduler, and Teradata Manager.

Additionally, the **tdwm** database was previously called **dbqrymgr**.



Levels of Workload Management

Three Tiers of Workload Management

Teradata Dynamic Workload Manager (TDWM)	Pre-Execution
Priority Scheduler	 Query Executes
Database Query Log	Post-Execution

TDWM

Control what and how much is allowed to begin execution.

Priority Scheduler

Manage the level of resources allocated to different priorities of executing work.

Database Query Log

Analyze query performance and behavior after completion.

What is TASM?

Teradata Active System Management (TASM) is made up of several products/tools that assist the DBA or application developer in defining and refining the rules that control the allocation of resources to workloads running on a system. These rules include filters, throttles, and “workload definitions”.

Rules to control the allocation of resources to workloads are effectively represented as workload definitions which are new with Teradata V2R6.1. Tools are also provided to monitor workloads in real time and to produce historical reports of resource utilization by workloads. By analyzing this information, the workload definitions can be adjusted to improve the allocation of system resources.

TASM is primarily comprised of three products to help create and manage “workload definitions”.

- Teradata Dynamic Workload Manager (TDWM)
- Teradata Manager
- Teradata Workload Analyzer (TWA)

Teradata Dynamic Workload Manager (known as TDWM or DWM) is a key supporting product component for TASM. The major functions performed by the DBA include:

- Define Filters and Throttles
- Define Workloads (new) and their operating periods, goals and PSF mapping/weights
- Define general TASM controls

The benefit of TASM is to automate the allocation of resources to workloads and to assist the DBA or application developer regarding system performance management. The benefits include:

- Fix and prevent problems before they happen. Seamlessly and automatically manage resource allocation; removes the need for constant setup and adjustment as workload conditions change.
- Improved reporting of both real-time and long-term trends – Service Level statistics are now reported for each workload. This helps manage Service Level Goals (SLG) and Service Level Agreements (SLA) – applications can be introduced with known response times
- Automated Exception Handling – queries that are running in an inappropriate manner can be automatically detected and corrected.
- Reduced total cost of ownership – one administrator can analyze, tune, and manage a system’s performance.



What is TASM?

What is TASM?

- Teradata Active System Management (TASM) is made up of several products/tools that assist the DBA or application developer in defining (and refining) the rules that control the allocation of resources to workloads running on a system.
- These rules include filters, throttles, and workload definitions.
 - **Workload definitions** are rules to control the allocation of resources to workloads.
- TASM is a new concept starting with Teradata V2R6.1.

Key products that are used to create and manage “workload definitions” are:

- Teradata Dynamic Workload Manager (enhanced with TASM)
- Teradata Manager (enhanced with TASM)
- Teradata Workload Analyzer (new with TASM)

The benefit of TASM is to automate the allocation of resources to workloads.

Teradata Dynamic Workload Manager (TDWM)

Teradata Dynamic Workload Manager (also known as Teradata DWM or TDWM) is a product that enables you to effectively manage the access to and utilization of a Teradata Database. TDWM provides both Query and Workload Management capabilities.

TDWM provides a graphical user interface (GUI) for creating rules that manage database access, increase database efficiency, and enhance workload capacity. TDWM consists of a server system component and a separate database with the Teradata Database called **tdwm**.

What is Query Management?

TDWM Query Management is a set of “rules” to determine whether logon and query requests will be accepted by the Teradata Database, and further to determine whether the execution of some query requests should be “delayed” (internally queued). The purpose of “delaying” queries is to limit the number of database resources that are tied up in processing low priority and/or long running queries. Queries that are delayed are still perceived as executing within the user’s session.

Why use a Query Management facility?

- Enables the DBA to effectively manage access to and the use of Teradata resources.
- Allows the processing of logon and query requests from all types of clients sources without any client software requirements.
- TDWM addresses the key problems of database system overload and network saturation that result from a large number of clients accessing the Teradata Database.

What is Workload Management?

Workload management on a system yields improved workload distribution and customized delegation of resources among the various workloads. A workload represents a portion of the queries that are running on a system. To use workload management in Teradata, a set of workload definitions must be established and enabled. A Workload Definition (WD) is a workload grouping and its operating rules to assist in managing queries. The requests that belong to the same workload will share the same resource priority and exception conditions.

Why use a Workload Management (new with TASM) facility?

- Assign queries to the correct workload before they start executing. Fix and prevent problems before they happen. Seamlessly and automatically manage resource allocation; removes the need for constant setup and adjustment as workload conditions change.
- A new application (i.e., Teradata Workload Analyzer) is also available to help in migrating existing environments to a “workload” environment.



Teradata Dynamic Workload Manager

TDWM provides a **Query Management (QM)** capability.

- A set of user-defined “**rules (or filters and throttles)**” is created to determine whether logon and query requests will be accepted by the Teradata Database.
 - These rules also determine whether the execution of some query requests should be “delayed” (internally queued).
 - TDWM utilizes a user named “**tdwm**” to store tables, views, macros, etc.
- Query Management provides “non-workload” **filters and throttles**:
 - **Filters** – object access and query resource rules used to **reject** queries
 - **Throttles** – object and load utility rules used to **delay** or **reject** queries

With TASM, TDWM also provides a **Workload Management (WM)** capability.

- A set of user-defined “**workload definitions**” is created to control the allocation of resources to workloads.
- Queries are associated with a “**workload**” based on who, where, and what criteria.

TDWM Query Management Architecture

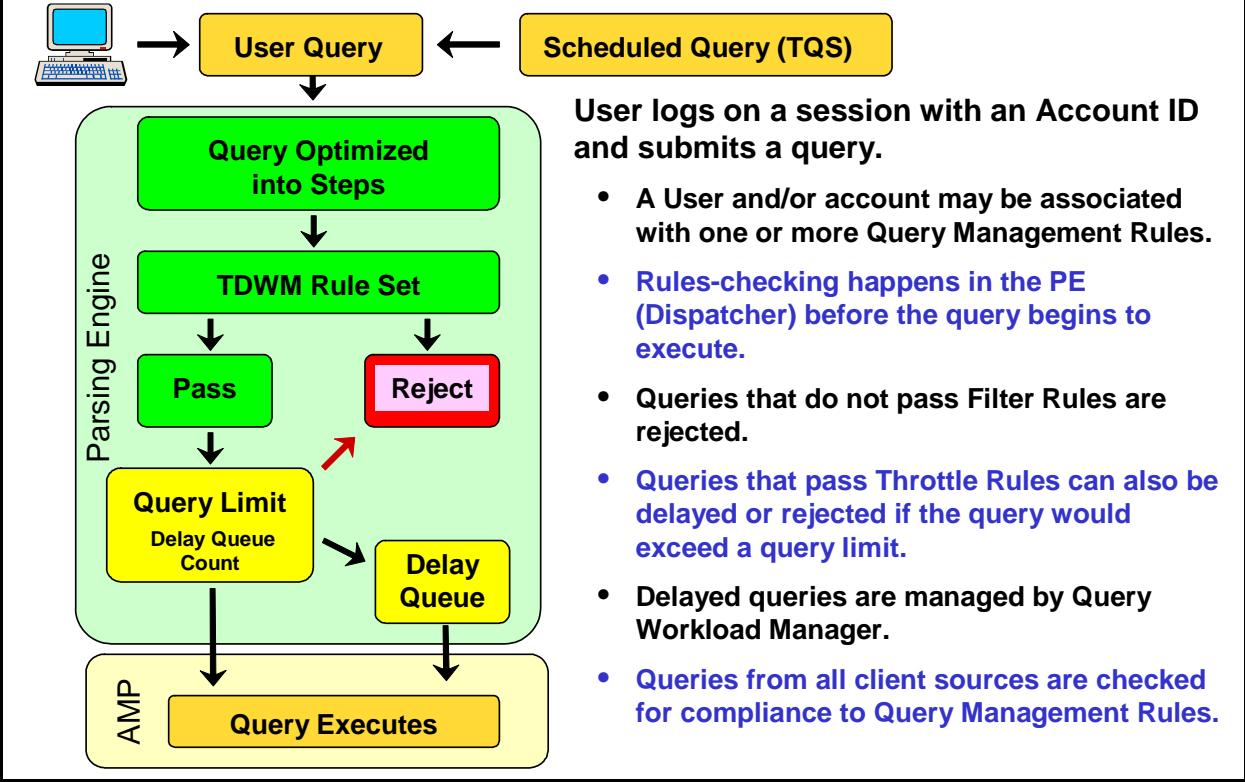
The TDWM rules you create are stored in tables in the Teradata database. Unless otherwise specified, every logon and every query in every Teradata Database session is checked against the enabled TDWM rules. That includes SQL queries from any supported Teradata Database interface, such as BTEQ, CLIV2, ODBC, and JDBC.

TDWM rules are loaded into the Dispatcher components of the Teradata Database. When a Teradata client application issues a request to the Teradata Database, the request is examined and checked by TDWM functions in the Dispatcher before being forwarded to the AMPs to execute the request against the user database.

The TDWM Query Management component examines database log on and query requests. It also analyzes the resource criteria of those requests and the objects it references. TDWM then compares the requests against the active rules to see if the requests should be accepted, rejected, or delayed



Query Management Architecture



Query Management Architecture (cont.)

The illustration on the facing page expands the rules shown in the previous illustration.

Query Management rules are classified as either Category 1, 2 or 3.

Query Management analyzes the incoming requests and compares the requests against the active rules to see if the requests should be accepted, rejected, or delayed.

- Queries that do not pass Category 1 Filter Rules are rejected
- Queries that do not pass Category 2 Throttle Rules can be delayed or rejected
- Queries that pass both Category 1 and Category 2 rules are checked against Category 3 Workload rules. Additional throttles can also be applied at the Workload Definition level.
- As queries execute within their assigned workload, they will be monitored against any exception rules.
- Violations of exception rules can invoke several actions from changing workloads, abort the query, send alert or run a program.

TASM provides two major capabilities that weren't available previously:

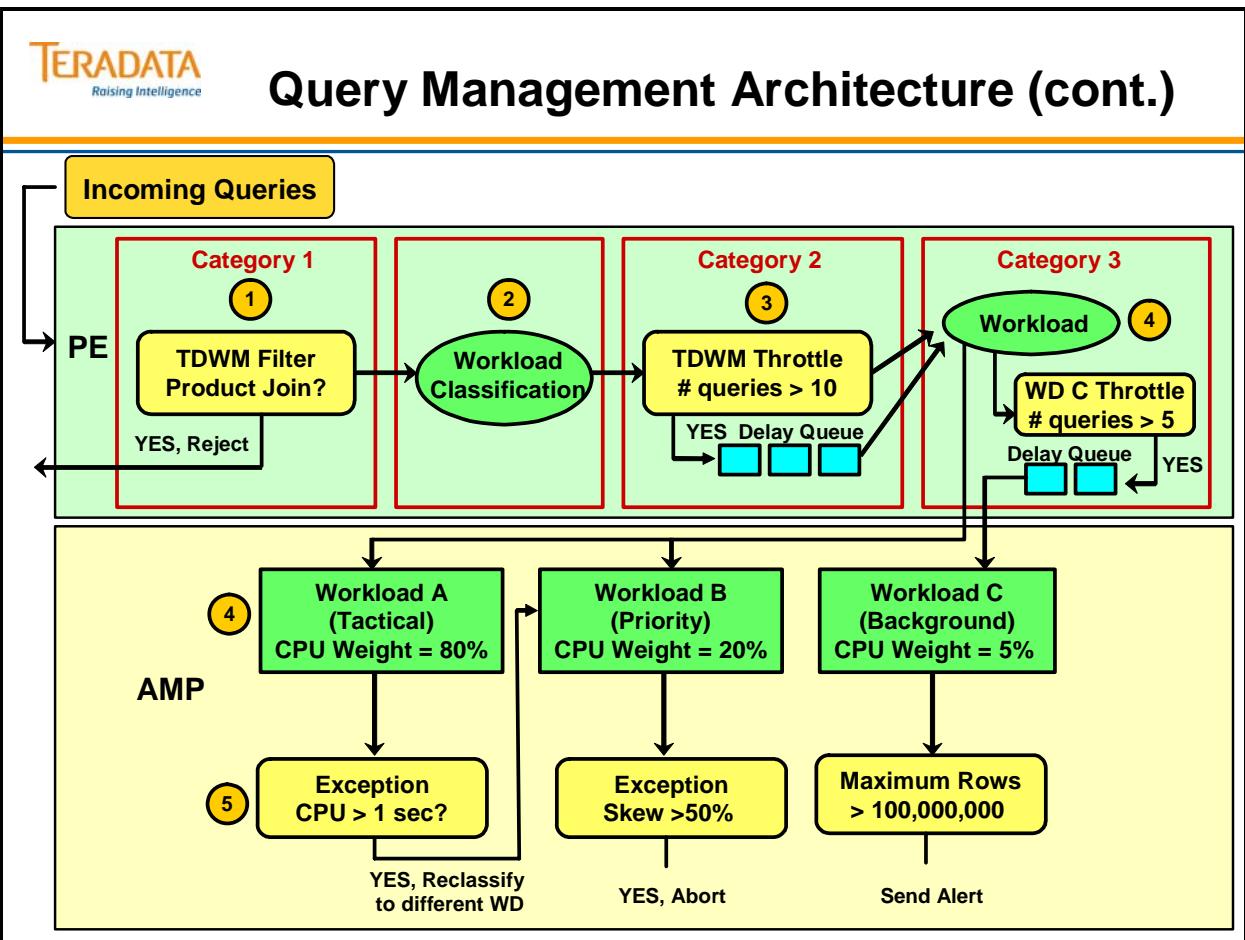
- With workloads, queries can be rejected, delayed or run in a performance group based on query attributes, not just the account string.
- Executing queries can be monitored and acted on. With previous releases, once a query started executing, TDQM (V2R5) or TDWM (V2R6.0) had nothing more to do with the query.

Starting with V2R6.1, TASM has the ability to monitor and manage running queries. This is done with exception criteria that is specified in the workload definition and the query with the criteria detected can have its priority changed (e.g., lowered) or even aborted.

How are queries associated with a workload?

- Teradata Parsing Engine software “classifies” a query into a Workload Definition (WD). It takes the query attributes (user name, account, optimizer estimates, etc.) and puts the query into the correct WD.

The example in the illustration shows five ways to control workload resource allocation.



TDWM Main Window – Types of Rules

TDWM is a graphical user interface (GUI) client utility that runs on Microsoft Windows. It allows a database administrator (DBA) to control the behavior of TDWM. The following are the main features of the Teradata DWM for query management:

- Create, delete, modify, view, enable, and disable filters, throttle, or workloads.
- Database Browser window for associating query objects with defined filters.
- Grant bypasses privileges to specific users, groups of users, or accounts.
- You can configure TDWM to affect how rules are enforced by ignoring EXPLAIN estimates that are below a specified level of confidence.
- Apply (or notify the database of) the latest filter, throttle, or workload changes.

The TDWM rules you create are stored in tables in the **tdwm** database. Unless otherwise specified, every logon and every query in every Teradata Database session is checked against the enabled TDWM rules in the **tdwm** database. That includes SQL queries from any supported Teradata Database interface, such as BTEQ, CLIV2, ODBC, and JDBC.

Note: Although every SQL request is subject to TDWM rules, you can set up specific users to bypass TDWM checking. These users are also called “unrestricted users”.

TDWM rules are loaded into the Dispatcher components of the Teradata Database. When a Teradata client application issues a request to the Teradata Database, the request is examined and checked by TDWM functions in the Dispatcher before being forwarded to the AMPs to execute the request against the user database.

There are three categories of workload management rules that are available. Any of these three categories can be enabled or disabled.

- Category 1: System-wide query management filters
- Category 2: System-wide query management throttles
- Category 3: Workload Definitions – new with TASM

Types of Query Management Rules

Object Access filters	Access to and from specific Teradata Database objects and object combinations by some or all users.
Query Resource filters	Which Teradata Database requirements are necessary to execute certain queries; such as limiting row count, processing time, or types of joins?
Object Throttles	How many sessions and/or queries can be running for specified Teradata Database objects?
Load Utilities	How many load utilities can be running on the Teradata Database either individually or collectively? Load utilities include FastLoad, MultiLoad, and FastExport.



TDWM Main Window – Types of Rules

Left pane is Rules Directory Information Tree (DIT).

From here, you can create, delete, enable, or disable rules.

Note the disabled rule.

The Settings option lets you define overall parameters for TDWM.

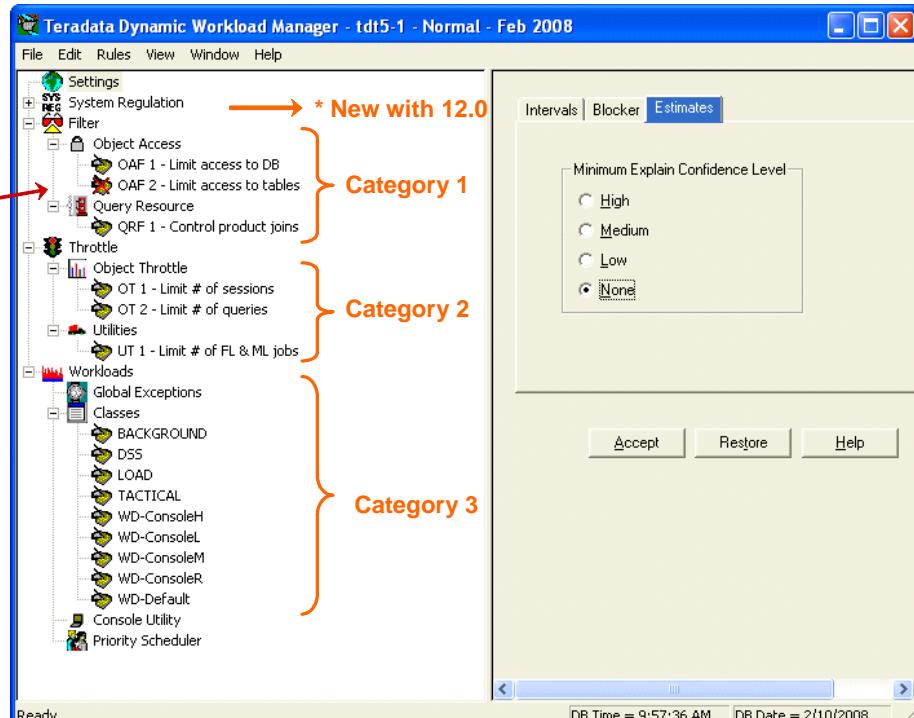
Types of rules include:

Category 1 – Filters

Category 2 – Throttles

Category 3 – Workload Definitions

* System Regulation – new TD 12.0.



Filters and Throttles for Query Management

The facing page identifies the 4 types of rules (2 filters and 2 throttles). The following is an overview of how TDWM processes logon and query requests to your Teradata Database.

1. Request is checked for any Context objects that are currently bypassed. If present, the request is immediately executed. For logon requests, go to step 5.
2. Teradata Optimizer step plan for each statement in the query request is traversed to determine the following:
 - Type of statement and type of step
 - Objects in the request
 - Estimated answer set size (number of rows) to be returned
 - Estimated number of rows involved in each step
 - Estimated total processing time required to complete execution
 - Table join required (product or unconstrained product)
 - Full table (all-rows) scan required
 - Types of steps which are all-AMP
 - Confidence level for each step
3. Step costs (that is, the row count and processing time estimates) are used only if the confidence level of the estimates is greater than or equal to the minimum confidence level you have configured.
4. Estimated resource usage values are compared with any global Query Resource rules.
5. Referenced Context and Query objects are checked against any applicable Filter rules. If any object is currently restricted, the step values for the request are compared to any Query Resource rules.
6. Referenced Context and Query objects are checked against any applicable Object Throttle rules. If any object is currently throttled, a supplementary indicator is returned so that the request is forwarded to the Query Manager task for throttle limit checking.
7. Object limits for a query are passed to the Workload Query Manager. The Workload Query Manager determines whether a logon is processed immediately or rejected, and whether a query request is processed immediately, delayed, or rejected.

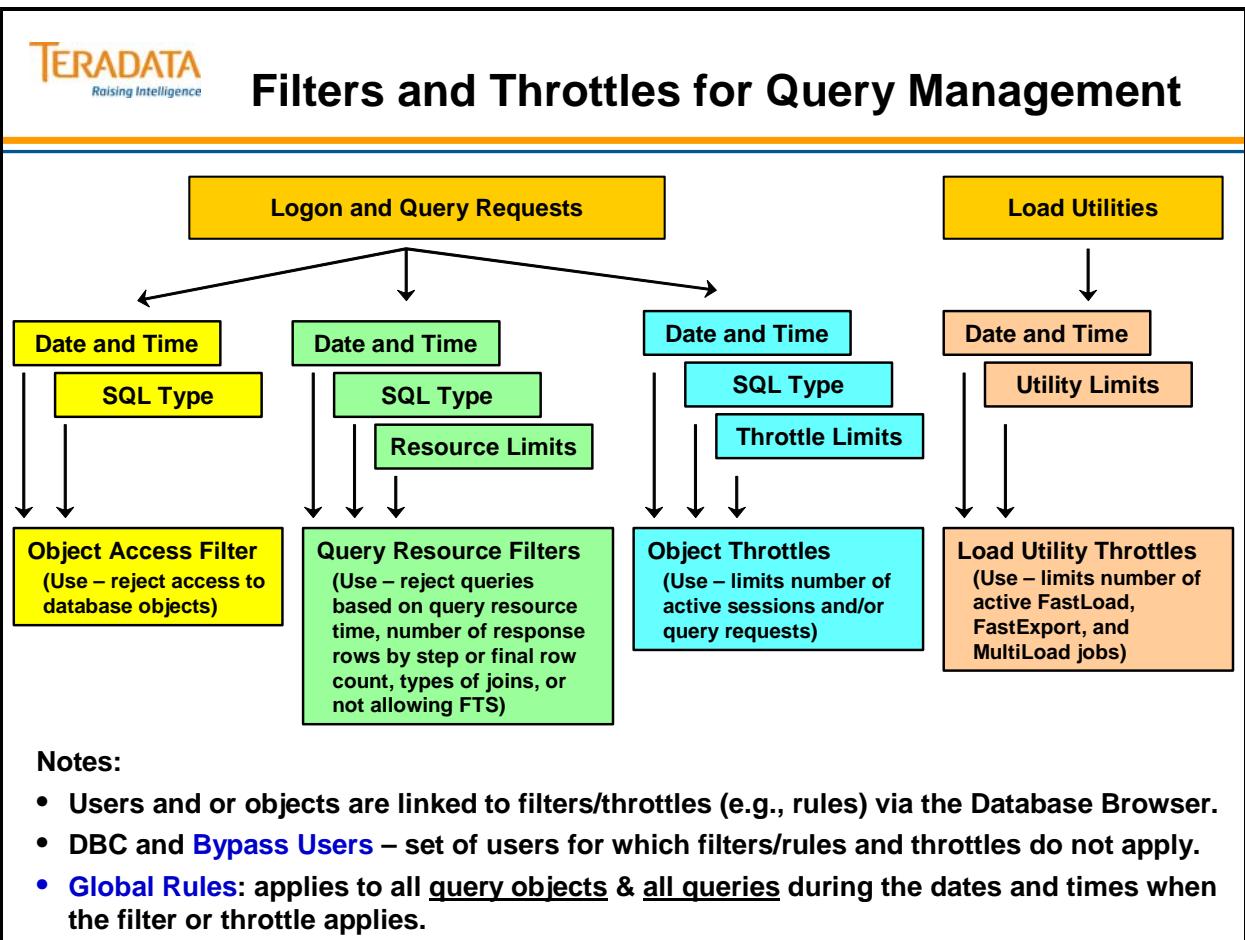
Note: Delayed objects are held until throttle limits allow them to run at which time an OK status is returned to the Teradata Database Dispatcher.

8. OK or reject status is returned to the Teradata Database Dispatcher.

Note: TDWM examines all logon and query requests in a SQL partition before they are sent to the Teradata Database for execution.

9. Teradata Database Dispatcher takes one of the following actions depending on the applicable rules:
 - Lets the request proceed to the AMPs.
 - Rejects the request or logon.

Note: Rejected logons and queries are stored in an exception cache. This cache is flushed based on the logging interval you define. Entries for rejected logons and queries are logged in the TDWM.EXCEPTIONLOG table so you can analyze them later.



Object Access and Query Resource Filters

Object Access filters are used to reject queries that attempt to access to all objects associated with the filter during the time period specified.

When you define an Object Access filter, you can specify that only combinations of issuing and query objects are restricted. This lets you selectively limit access to the Teradata Database, tables, macros, and so on. For example, you could create a filter that never allows specific users access to specific database tables.

Defining Query Resource filters lets you reject queries based on database resource usage for any issuing object, query object, or object combinations associated with this type of filter. You define how resource usage is limited, as well as the dates and times the resource usage limits apply.

You can configure how Query Resource filters are enforced by ignoring EXPLAIN estimates below a specified level of confidence. For example, if the row count estimate on a query is generated with “no confidence” and the minimum explain confidence level is set to low confidence, then the row count estimate is not used.

Example of Object Access filters include:

- On Saturday and Sunday, user A cannot log on to the Teradata Database.
- On Weekdays between 8:00 am and 5:00 pm, table B cannot be accessed.

Examples of Query Resource filters include:

- On Fridays between 9:00 am and 2:00 pm, table B cannot be involved in a Product Join AND that returns more than 1 million rows.
- On Tuesdays between 12:30 pm and 4:00 pm, queries estimated to take longer than 30 minutes cannot run.

SQL Types

For Object Access filters, Query Resource filters, and Object throttles, you can specify the types of SQL requests to which the rule applies. For example, you can specify ALL, DDL, DML, or SELECT.

Global

Global filters apply to all objects, and as a result to all logon and/or query requests during the specified time period. If a filter applies to all objects, you can specify it as a global rule. Because a global rule automatically applies to all Teradata Database objects, you do not need to associate individual Teradata Database objects with the rule.

Caution: Defining a global Object Access rule causes **all** of the specified statement type requests to be rejected except those from the DBC user and any bypassed objects.



Object Access and Query Resource Filters

Object Access Filters

Object Access Filters **reject** any access to database objects that you associate with the restriction.

Example:

If you associate a table (T1) with an access restriction for a group of users, then TDWM rejects any query that contains T1 within the defined operating environment (may be defined time period or a defined state such as LOAD).

Query Resource Filters

Query Resource Filters **reject** any access to database objects based on resource usage limits, as well as the dates/times or states that the resource usage limits apply.

Example:

On weekdays between 08:00 and 17:00, queries estimated to take longer than 30 minutes are not allowed to execute for users assigned to a specific performance group.

You can configure how Query Resource filters are enforced by ignoring EXPLAIN estimates below a specified level of confidence.

Object and Load Utility Throttles

For Object Throttles, you define additional limits on logon and queries requests. For Load Utility Throttles, you choose the type of load utility to which the rule applies instead of the type of SQL request.

Object Throttles

Defining Object Throttles lets you limit the number of logon sessions and/or queries active on for particular Teradata Database objects. You can define Object Throttles that apply to most types of Teradata Database objects. You cannot associate Object Combinations with Object Throttles. You can associate Macros and Stored Procedures with Object Throttles. However, they are treated like table objects in that we do not know that they are a “Macro” or a “Stored Procedure” per se. TDWM will just know that the name in the rule matches the name on the object list.

You can set up this type of rule to reject or to delay any query that cannot be immediately processed. If more than one Object throttle applies to an object, the one with the lowest limit one is used.

Note: SQL requests evaluated under this category must include an **ALL-AMP** step to be considered against throttle values. Single AMP operations (for example, prime index) are always allowed to run and are not counted against throttle limits on context objects.

Object Throttles were referred to as Workload Limits in TDQM (Teradata V2R5.0 and 5.1).

Load Utility Throttles

Defining Load Utility throttles lets you control how many load utilities are simultaneously running on a Teradata Database at any given time. Using this throttle type lets you override the MaxLoadTasks value set using the DBS Control Utility. Setting a throttling rate lets you override the value without having to change it using the DBS Control Utility.

You can specify limits for all load utilities as a group, and/or specify limits for each individual load utility. Because Load Utility throttles apply only to the kind and number of load utilities running on the Teradata Database, you cannot associate Teradata Database objects with them.

This option applies to the FastLoad, FastExport, and MultiLoad utilities.

Context Objects

Context objects relate to the conditions in which a request is issued. Because they relate to who issued the request, they are also called **who** objects. The types of context objects you can associate with rules are Users, Accounts, Performance Groups, and Profiles.



Object and Load Utility Throttles

Object Throttles

Object Throttles limit the number of logon sessions and/or active queries.

Example:

On Weekdays between 8:00 and 17:00, performance group \$M cannot have more than 200 simultaneous sessions on the entire Teradata Database.

On Weekends, performance group "\$H" cannot have more than 5 simultaneous queries on the entire Teradata Database – delay queries, do not reject.

Load Utility Throttles

Load Utility Throttles allow you to control how many load utilities are simultaneously running on a Teradata Database at any given time.

Example:

On Weekdays between 8:00 and 17:00, the maximum number of simultaneous FastLoad and/or MultiLoad jobs is 3.

TDWM – General Settings

Minimum Explain Confidence Level

You can configure how Query Resource rules are enforced by ignoring EXPLAIN estimates that are below a specified level of confidence. To set the minimum explain confidence level; choose high, medium (index), low, or none. The default is none.

Minimum Explain Confidence Level impacts how Query Resource filters, Object Throttles, or Workload Definitions with step time threshold are enforced.

- Query Resource filters depend on optimizer estimates to determine if a query will execute immediately or be rejected.
- Statistics collection becomes even more important for ensuring optimal performance.
- This option causes DWM to bypass any query step from consideration, if the step's confidence level is *BELOW* the specified confidence level.
- For example, if you choose the Medium confidence level, then any query step that had Low or No confidence would be bypassed by the rule.
- Only steps meeting the minimum level would be summed when assessing the Maximum Processing Time and Final Row Count.

Deadlock Settings

The Deadlocks or Blocker tab (not shown in detail) allows you to set controls for the handling deadlock situations that can occur when individual requests within multi-request transactions are delayed through the use of concurrency throttles. Valid values are zero through three. Zero indicates that no deadlock detection is used.

Interval Settings

The Intervals tab (not shown in detail) allows you to set exception and logging intervals.

In the Dashboard Data Interval box, specify how often Summary data is saved and reset for use by the Teradata Manager dashboard. Valid values are 1 to 600 seconds. Dashboard reports are refreshed every 60 seconds.

In the Event Interval box, specify how often Teradata Database asynchronously checks for current time and system condition values.

In the Exception Interval box, specify how often the Teradata Database asynchronously checks queries executing within the current set of workload classes to see if they meet the relevant workload class exception criteria. Valid values are 1 to 3600 seconds; default is 600 seconds.

In the Logging Interval box, specify how often Exception, Summary, and Detail Log table entries are generated from the memory cache where the data is collected. Valid values are 60 to 3600 seconds; default is 600 seconds. This interval affects when statistical/logging data is available in the DBQL/TDWM log tables.



TDWM – General Settings

Minimum Explain Confidence Level impacts how Query Resource Filters, Object Throttles, or Workload Definitions are enforced.

- TDWM depends on optimizer estimates.
- TDWM will bypass any query step if the step's confidence level is **BELOW** the specified confidence level.

For example, if you choose Low confidence level, then any query step with No confidence is bypassed.

<input type="radio"/> High <input type="radio"/> Medium <input type="radio"/> Low <input checked="" type="radio"/> None
<input type="button" value="Accept"/> <input type="button" value="Restore"/> <input type="button" value="Help"/>

Intervals:

- **Dashboard** – how frequently to refresh TM Dashboard.
- **Event Interval** – specify how often Teradata asynchronously checks for current time and system condition values.
- **Exception** – used for workload exceptions.
- **Logging** – how frequently to flush workload collections to disk.

Dashboard Data Interval: <input type="text" value="5"/> seconds Event Interval: <input type="text" value="180"/> seconds Exception Interval: <input type="text" value="600"/> seconds Logging Interval: <input type="text" value="600"/> seconds
<input type="button" value="Accept"/> <input type="button" value="Restore"/> <input type="button" value="Help"/>

Object Access Filter Properties

When you create a new Filter or Throttle rule using DWM, there are three common properties you need to define:

- Descriptive attributes (Name, Description, etc.)
- A system period-the dates and time when the rule is in effect
- The types of SQL requests to which the rule applies (not available for Load Utility Throttles)

To create a new Object Access Filter:

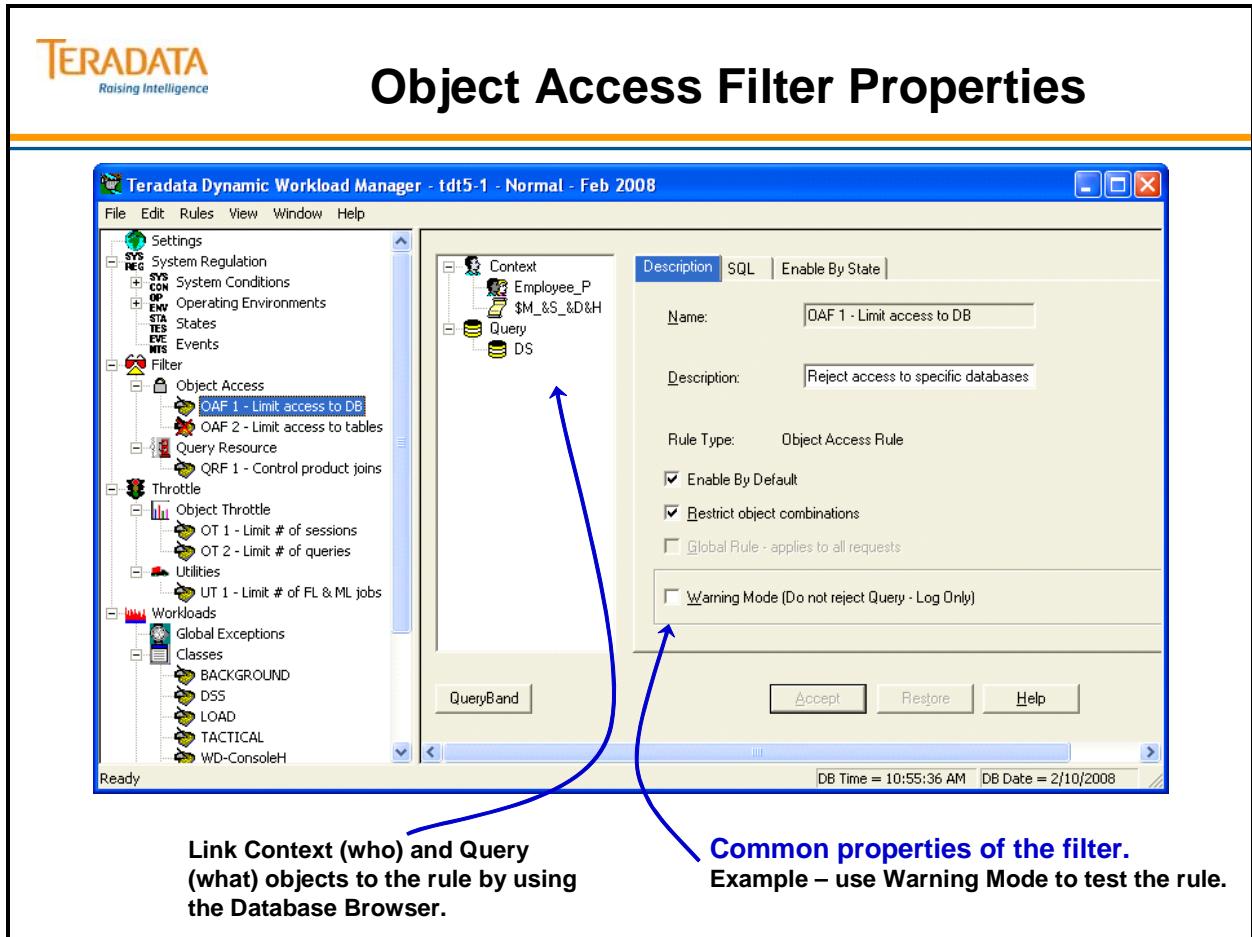
1. Enter the rule name, up to 30 characters, in the Name text box.
2. Enter a brief description of the rule, up to 50 characters, in the Description text box.

Following are the properties and parameters for creating the rule:

Restrict object combinations	Select this check box when you want to restrict combinations of issuing objects with query objects.
Global Rule (applies to all requests)	Select this when the rule should be a global rule that is applied to all logon and query requests. This rule restricts all users except DBC and any bypassed users.
Warning Mode (Rejected Queries Logged)	Select this check box if you want the rule only to log rejected queries.

Notes:

- If a filter applies to all objects, you can specify it as a global rule. Because a global rule automatically applies to all Teradata Database objects, you do not need to associate individual Teradata Database objects with the rule. Defining a global Object Access rule causes **all** of the specified statement type requests to be rejected except those from the DBC user and any bypassed users.
- If you want to see the effects of a filter before you permanently enable it, you can define it as operating in a warning mode. When a rule is in a warning mode, the following events occur:
 - Queries are evaluated as if the rule is in normal mode
 - Errors are logged only for queries that would potentially be rejected
 - A error status code or message is **not** returned to the end-user



Query Resource Filter Properties

This chart provides information on Query Resource controls and how to use them.

Use this control...	To...
Maximum Row Count box	<p>Specify the maximum estimated row count allowed for any step in a request.</p> <p>This is the maximum row count estimate during any step in the query. The default value is zero which means the row count estimate is not used in processing the rule.</p> <p>Note: You define the minimum level of confidence you want to use in the estimates in the Query Management tab of the Settings dialog box.</p>
Final Row Count box	Specify the final row count limit for a query. This sets the maximum estimated number of rows that can be returned by a query request.
Rows list	<p>Define the row count as one of the following values:</p> <ul style="list-style-type: none"> <i>Rows</i> <i>Thousand Rows</i> <i>Million Rows</i> <i>Billion Rows</i>
Maximum Processing Time box	Enter the hours, minutes, and seconds you estimate as the greatest amount of time a query is allowed to run.
Restricted Table Joins list	<p>Select the type of table joins you want to restrict from one of the following items:</p> <ul style="list-style-type: none"> <i>None</i> <i>All Joins</i> <i>Product Joins</i> <i>Unconstrained Product Joins</i> <p>Select <i>None</i> when you want to allow all joins.</p>
No Full Table Scans Allowed check box	Restrict full table (all row) scans from occurring while a query is running. Clear the check box if you want full table (all row) scans to occur while a query is running.
Apply Properties frame	
OR option	Specify that the rule is triggered by any one of the specified conditions.
AND option	Specify that the rule is triggered only when all of the specified conditions are met.

TERADATA
Raising Intelligence

Query Resource Filter Properties

Limits

Conditions

Maximum Row Count: 0 Rows
 Final Row Count: 100 Million Rows
 Maximum Processing Time: 5:00 (HHH:MM:SS.dd)
 Restricted Table Joins: Unconstrained Product Join
 No Full Table Scans Allowed:

Apply Properties: OR AND

Accept | Restore | Help

DB Time = 10:59:37 AM DB Date = 2/10/2008

OR option – the rule is triggered by any one of the specified limits/conditions.
AND option – the rule is triggered only when all of the specified limits/conditions are met.
 Note: Join/Scan and Row limits with AND property must occur in the same step rather than anywhere in the request.

Object Throttling Properties

This chart provides information on Object Throttling controls and how to use them.

Use this control...	To...	Comments...
Enable Session Limit check box	Select this check box to limit the number of logon sessions. Then enter the maximum number of logon sessions for this rule.	
Enable Query Limit check box	Select this check box to limit the number of simultaneous queries with all-AMP data manipulation steps. Then enter the maximum number of simultaneous queries with all-AMP data manipulation steps that can be performed by a user, account name, performance group, or users within a performance group.	Warning: Defining workload limits of two or less can set up deadlocks between locks being needed and requests being held by DWM.
Step Time Threshold box	<p>Specify the step time threshold for all-AMP data manipulation steps. If a query does not have any all-AMP data manipulation steps estimated to exceed this threshold, then DWM does not limit (count, reject, or delay) the query.</p> <p>If you specify a time value of zero, DWM limits (count (warn), reject, or delay) all all-AMP data manipulation steps.</p>	<p>If the number of queries is less than the number you indicate in the Query Limit property, the query runs immediately.</p> <p>However, if the query limit is reached, DWM checks the Reject Queries property and the query is either rejected or delayed as follows:</p> <p><u>Rejected</u>: If you selected the Reject Queries check box, DWM rejects the query and returns an error to the user.</p> <p><u>Delayed</u>: If you did not select the Reject Queries check box, DWM puts the query in the queue. When the number of running queries falls below the defined throttle limit, queries in the queue are submitted until the throttle limit is reached.</p>
Reject Queries (Default is Delay) check box	Select this check box if you want all queries that exceed the query limit to be rejected. Clear this check box if you want these queries to be delayed	
Disable Object Throttling Rule Override check box	Specify that the DBA cannot override (abort or execute) any queries delayed by this rule.	Caution: Queries affected by this override cannot be manually manipulated by the DBA. For example, when Replication Services dynamically creates an Object Throttle rule against a replicated table that should not be accessed, the DBA should not be allowed to inadvertently initiate a query against the table.

The screenshot shows the Teradata Dynamic Workload Manager interface. On the left is a tree view of settings, including System Regulation, System Conditions, Operating Environments, Filter, Object Access, Query Resource, Throttle, Utilities, Workloads, and Global Exceptions. Under Throttle, Object Throttle is selected, showing two rules: OT 1 - Limit # of sessions and OT 2 - Limit # of queries. The main window displays the 'Object Throttling Limits' tab of the 'Object Throttling Properties' dialog. It has sections for 'Description', 'SQL', 'Object Throttling Limits', and 'Values By State'. The 'Object Throttling Limits' section contains checkboxes for 'Enable Session Limit' (unchecked) and 'Enable Query Limit' (checked). A red box highlights this checked state with the text: 'If used, sessions will not be delayed.' Below it is a 'Query Limits' section with 'Maximum Queries' set to 10 and 'Only limit queries with Step Time Threshold' set to 1:00 (HHH:MM:SS.dd). There are two radio buttons: 'Delay Query' (selected) and 'Reject Query'. Another red box highlights the 'Delay Query' option with the text: 'Disables the ability to execute or abort queries in the Delay Queue.' At the bottom are 'Accept', 'Restore', and 'Help' buttons, and a status bar showing 'DB Time = 11:03:36 AM DB Date = 2/10/2008'.

Object Throttles – used to control the number of queries or sessions for specific users, accounts, performance groups, or profiles.

Linking Objects to Rules

When you associate objects, you need to link only those Teradata Database objects you want to restrict. Generally, you link one or several Teradata Database objects with a TDWM rule. However, you cannot link some objects to certain types of rules; some limitations apply. For example, you can link performance groups only with Object Throttle rules.

When you link both context and query objects with an Object Access or Query Resource filter for which the **Restrict object combinations** check box was selected, the rule applies to all the combinations of issuing and query objects. If the **Restrict object combinations** check box was not selected, the rule applies to each associated object separately.

Unlinking Objects from a Rule

You can disassociate (unlink) Teradata Database objects from a rule whenever you like. When you unlink an object from a rule, the object is not deleted. The object still exists, but that particular rule no longer applies to the object.

Objects already linked to a selected rule appear in **Objects** DIT, the frame to the left of the tabs in the **Properties** pane.

To unlink an object from a rule,

1. In the **Objects** DIT, select the object type you want to unlink.
2. Use one of the following options:
 - From Teradata Dynamic Workload Manager, open the Rules menu and choose Filter or Throttles. Next, choose Delete, and then choose Delete Object.
 - Right-click the object to open a shortcut menu, and then choose Delete.
 - Press the Delete key.
3. A confirm deletion dialog box opens and click Yes to confirm.

Saving and Applying the TDWM Rule Set

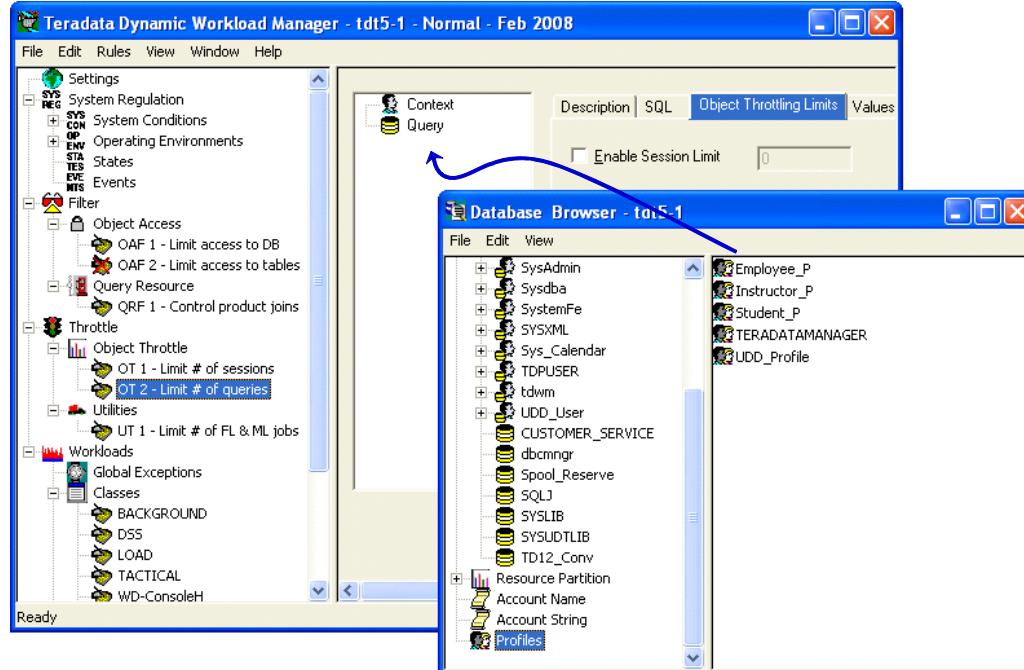
After you create and enable individual filters and throttles, you need to save them to the **tdwm** database. From the Rules menu, choose Rule Sets, and then select Save. The rules you created are saved as the DWM rule set in the **tdwm** database.

You also need to load the data in the **tdwm** database into the Dispatcher components of the Teradata Database. From the Activate Rule Set dialog box, you can select the specific categories of changes you want to apply. This lets you apply changes to your rule set to the Teradata Database in phases.

Option: You can also activate each TDWM rule category separately from the others to achieve the appropriate level of workload management and avoid incurring unnecessary overhead.



Linking Objects to Rules



The **Database Browser** lets you link objects to rules by dragging the object onto the rule.

Workload Definitions

A workload represents a portion of the queries that are running on a system. A Workload Definition (WD) is a workload grouping and its operating rules to assist in managing queries. The requests that belong to the same workload will share the same resource priority and exception conditions. It consists of:

- Classification Criteria: criteria to determine which queries belong to the workload. This criteria defines characteristics which are detectable prior to query execution. This is also known as the "*who*", "*where*", and "*what*" criteria of a query. For example, "*who*" may be an account name, "*where*" is the database tables being accessed, and "*what*" may be the type of statement (UPDATE) being executed.
- Exception Criteria: criteria to specify "abnormal" behavior for queries in this workload. These criteria are only detectable after a query has begun execution. If the exception criteria are met, the request is subject to the specified exception action which may be to lower the priority or abort the query.
- Operating Periods: a description of hours of the day and/or days of the week (or month). Directives may be specified for exception handling and Priority Scheduler settings can be changed for each operating period.

A Workload Definition is mapped to an Allocation Group (AG) of Priority Scheduler.

Why Create Workload Definitions?

The reason to create workload definitions is to allow TASM to manage and monitor the work executing on a system.

There are three basic reasons for grouping requests into a workload definition.

- Improved Control – some requests need to obtain higher priority to system resources than others. Resource priority is given on the basis of belonging to a particular workload.
- Accounting Granularity – workload definitions allow you to see who is using the system and how much of the various system resources. This is useful information for performance tuning efforts.
- Automatic Exception Handling – queries can be checked for exceptions while they are executing, and if an exception occurs, a user-defined action can be triggered.



Workload Definitions

What is a Workload Definition?

- It is a description of rules that represent **who**, **where**, and **what** of a workload. A Workload Definition is assigned to a Priority Scheduler allocation group.

Why Create Workload Definitions?

- **Improved Control of Resource Allocation** – resource priority is given on the basis of belonging to a particular workload.
 - Classification rules permit queries to run at the correct priority from the start.
- **Improved Reporting** – workload definitions allow you to see who is using the system and how much of the various system resources.
 - Service level statistics are reported for each workload.
 - Real-time and long-term trends for workloads are available.
- **Automatic Exception Handling**
 - After a query has started executing, a query that is running in an inappropriate manner can be automatically detected. Actions can be taken based on exception criteria that has been defined for the workload.

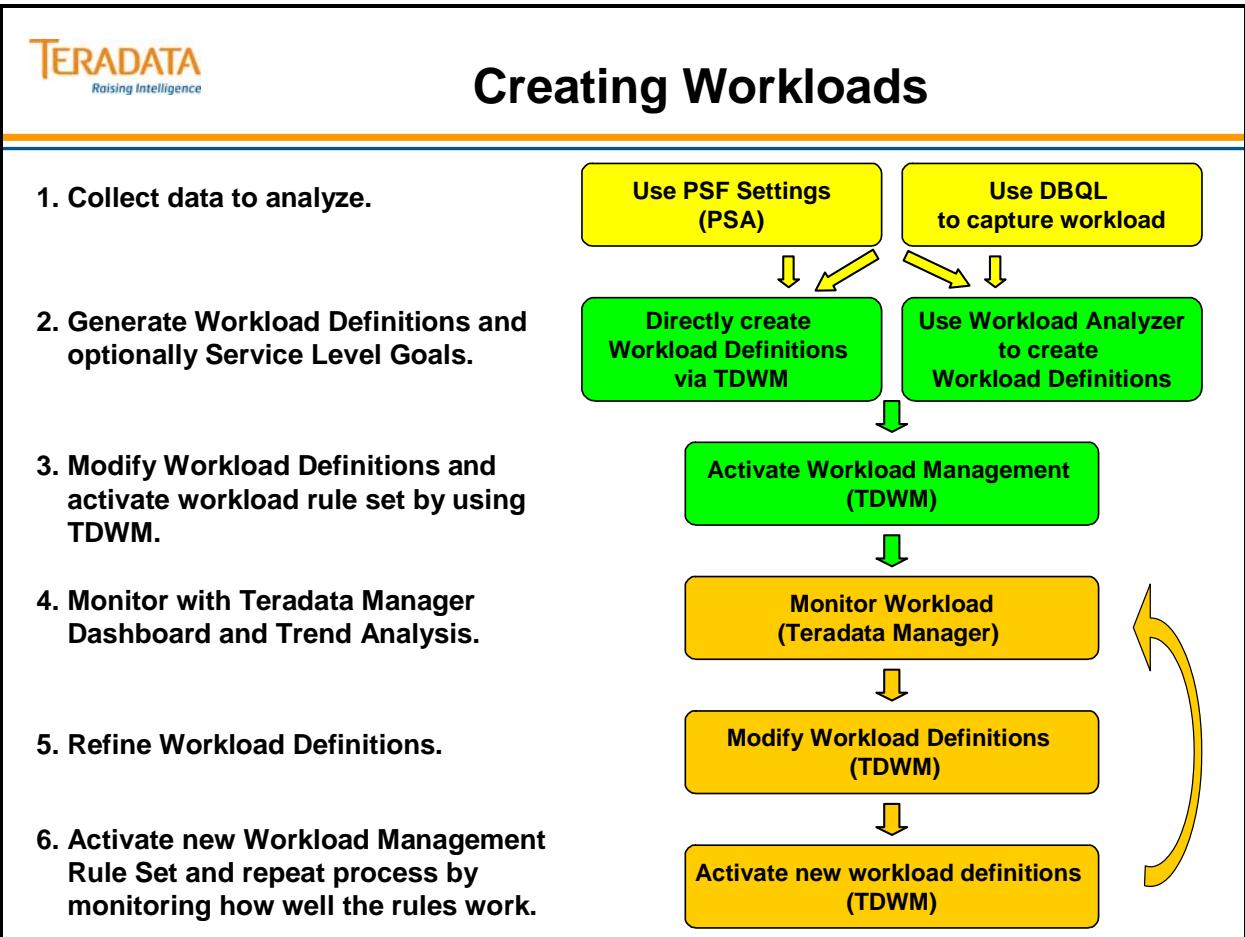
Creating Workloads

The facing page provides a list of the major tasks that are used in TASM. A brief description of each of the tasks is also included.

1. **Collect data to analyze** – one source of data is Priority Scheduler settings captured in PD (Priority Definition) sets. The second source is DBQL log data that represents captured queries for a period of time (e.g., three months) that represent the typical workload.
2. **Generate Recommended Workload Definitions** – use/combine the information from the two sources to generate the initial workload definitions. There are primarily two techniques in which workload definitions can be created:
 - Use Teradata Dynamic Workload Manager to directly create workload definitions from scratch. In doing so, users first collect query log information for the existing workload mix.
 - Use Teradata Workload Analyzer to analyze and create workload definitions based on the two sources identified in step #1.
3. **Activate the Workload Management Rule set** – the Teradata Dynamic Workload Manager (TDWM) administrator is used to optionally modify the workload definitions and activate them on the system.

Steps 4 – 6 are a reiterative process.

4. **Monitor the Workload** – Teradata Manager has new features which allow the administrator to monitor workloads. These enhancements are part of the Dashboard and Trend Analysis displays.
5. **Refine Workload Definitions** - based on how well the workload definitions are working, they may need to be adjusted via TDWM.
6. **Activate “new” Workload Definitions** – use TDWM to activate the adjusted workload definitions.

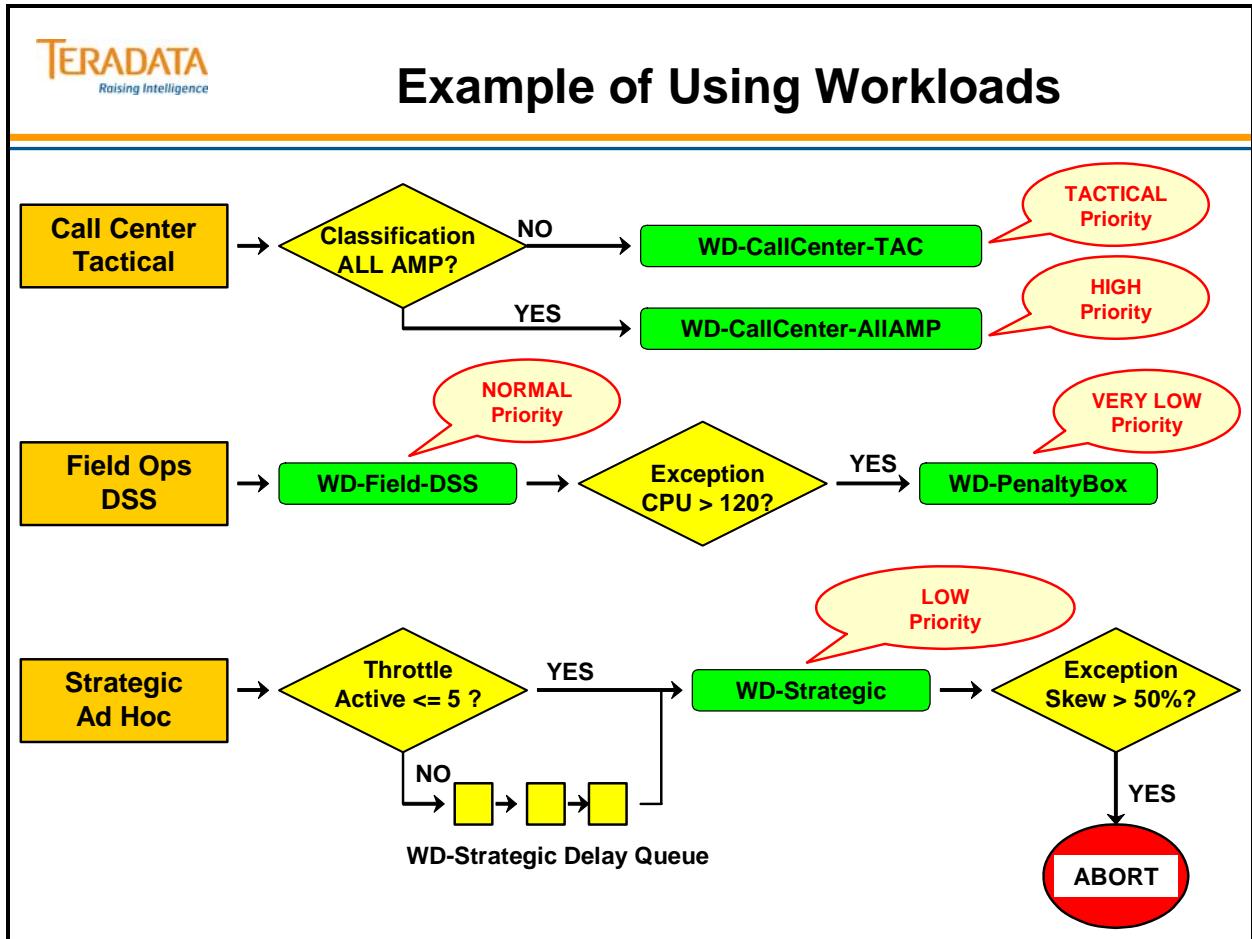


Example of Using Workloads

The facing page illustrates an example of creating five workload definitions to handle a mix of queries.

Recommendations Summary When Defining Workload Criteria

- Lead with “Who” criteria, and add “Where”, “What” and exception criteria only when necessary.
- “Who” criteria is the most exact and has the least overhead.
- Keep the total number of workloads small (e.g. 10-20)
- Keep the number of criteria associated with a workload as simple as possible, avoiding long confusing lists of and’d classification and exception criteria as well as include/exclude lists of “Who”/“Where” criteria.
- Use order of evaluation to put more specific definitions ahead of less specific definitions, enabling the system to accurately classify a request without having to evaluate the request against the entire list of workloads and their criteria.
- Avoid exception monitoring if possible, relying on classification to properly assign a request before execution even begins. This enables concurrency throttles if necessary as well as avoiding even a momentary mis-classification of the request in the unintended priority resource allocation.
- If using exception monitoring, use the longest exception interval as possible to keep exception monitoring overhead lower, yet still meet your goals for detecting exceptions in a timely manner. If no exception monitoring is used, set the exception interval to the maximum 3600 seconds.



TDWM – Create Workload Definitions

Teradata Dynamic Workload Manager (TDWM) can be used to directly create Workload Definitions or may be used to refine workloads that have been created via the Teradata Workload Analyzer.

To directly create a new workload definition, use TDWM and right-click on Workload and select New Definition. Select a name for the workload and complete the classification criteria. After creating and/or refining workloads, remember to activate them via TDWM.

Workload attributes that can be specified include:

- Name – the name of the workload (up to 30 characters)
- Description – a description of the workload (optional – up to 50 characters)
- Logging Mode – the level of logging for this workload:
 - None – no logging of WD activity. Teradata Manager Trend Analysis can't provide a report if this choice is taken (because there is no logging).
 - Workload Summary – periodic logging of summary statistics to a summary log table (DBC.TDWMSummaryLog) based on a specified interval.
 - Query Detail – writes “default row” to the DBQL table (DBC.DBQLLogTbl). If you also have created rules within DBQL to write detail rows (e.g., full SQL), rows will not be written twice to the DBQL tables.
- Enforcement Priority – specifies the conceptual importance of the workload.
 - **Tactical** – short queries with a fixed response time requirement.
 - Priority – important queries that should get extra resources.
 - Normal – normal queries.
 - Background – low priority queries with no response time requirement.

A Workload is assigned an Enforcement Priority. A Workload can only be mapped (shown in a later step) to an Allocation Group (AG) that has the same Enforcement Priority. When an Allocation Group (AG) is created, it is also assigned an Enforcement Priority. Certain features are restricted so that they are only available to workloads with Tactical Enforcement Priority. These features include the “expedited” flag that allows reserved AWTs and the ability to use PSF milestones.

Enforcement Priority is a feature that will have its functionality expanded in a future release. In a future release, if the system is too busy for the tactical queries to meet their Service Level Goals (SLG), then the Teradata Database software will throttle back the work into a different enforcement priority.



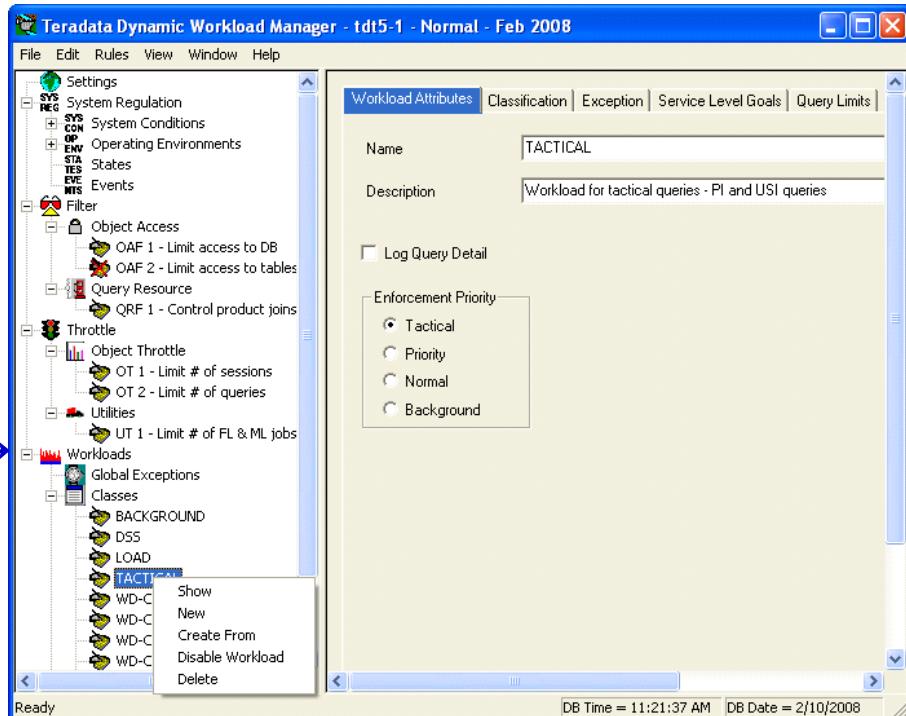
TDWM – Create Workload Definitions

Workloads provide the TASM capability.

Log Query Detail – writes “default row” to DBQLogTbl.

Enforcement Priority:

A Workload can only be mapped (later step) to an Allocation Group (AG) that has the same Enforcement Priority.



WD – Classification Criteria

After specifying a new workload definition name and attributes and clicking on the NEXT button, it is necessary to define classification criteria for the workload.

The basic classification criteria describes the "who", "where", and the "what" of a query. This information effectively determines which queries will run in a workload.

You can specify up to six different criteria for a workload. A query is classified into a Workload Definition (WD) if it satisfies all of the Classification criteria. Normally, you will only need to specify one or two criteria for a workload definition.

The "who" criteria define who is executing the query. Examples include:

- **Account** – the user's unexpanded account string (e.g., \$TAC_H\$&S&D&H)
- **User** – the Teradata username (e.g., NomarJoe, SmithRobert, DBC)
- **Client ID** – the logon name on the network client (e.g., JN450824)
- **Client Address** – the IP address of the network client (e.g., 141.206.28.51)
- **Profile** – the user's Teradata profile name (e.g., Buyer)
- **Application** – the application name on the network client (e.g., QUERYMAN)

Avoid long include/exclude lists associated with "Who" and "Where" criteria. Consider the use of accounts (that combine many users into one logical group) or profiles to minimize long "Who" include/exclude lists.

The "where" criteria defines which database objects are referenced by the query.

- **Data Objects** – choices include databases, tables, views, macros, and stored procedures. Note that UDFs are not supported.

The "what" criteria for the query is based on optimizer estimates. Options include:

- **AMP Limits** – is this an all-AMP request or not? Selecting this checkbox causes the workload to accept only queries that are *not* all-AMP queries.
- **Load Utility Type** – FASTLOAD, MULTILOAD, FASTEXPORT (or all three) When selecting additional criteria, be aware that you cannot combine a load utility type with anything other than a "who" criteria.
- **Statement Type** – the type of statement being submitted (e.g., SELECT, DDL, DML)
- **Row Count** – minimum and/or maximum rows at each step for spool files and result set
- **Final Row Count** – minimum and/or maximum rows for result set only
- **CPU Time** – minimum and/or maximum estimated processing time. You can specify CPU time in hundredths of a second (using the format HHH:MM:SS.dd).

"Who" criteria has lower overhead than "where" and "what" because "who" is determined once per session logon, whereas "where" and "what" are determined once per query.



WD – Classification Criteria

- Workload definitions have classification criteria that specify the **WHO**, **WHERE**, and **WHAT**.
 - **WHO** – determines who is executing the query
 - For example: Account string, Username, Profile, Application name, etc.
 - **WHERE** – defines which database objects that are referenced by the query
 - For example: databases, tables, views, macros, and stored procedures , etc.
 - **WHAT** – is based on optimizer estimates
 - For example: Number of AMPs used, step row count, final row count, CPU time, etc.
- A query will run in a specific workload based on the classification criteria.
 - With previous Teradata releases, queries were only classified by account string.
- Each of the criteria are ANDed together.
 - Normally, you only need to have 1 or 2 classification criteria (maximum of 6).
 - A query is classified into a Workload definition (WD) if it satisfies all of the Classification criteria.

WD – Exception Criteria

Optionally, you can define the Exception Criteria that are used to set thresholds for resource utilization. You can define one or more of the following exception criteria:

Unqualified Criteria: The following criteria are effectively “ANDed” together. If all of the specified criteria are met, then the exception action is triggered.

- **Maximum Rows** – the maximum number of rows in a spool file or final result.
- **IO Count** – the maximum number of disk I/O's performed on behalf of the query.
- **Spool Size** – the maximum size of a spool file.
- **Blocked Time** – the length of time the query is blocked by another query.
- **Elapsed Time** – the length of time the query has been running.
- **Number of AMPs** – the number of AMPs that participate in the query.
- **CPU Time** – the maximum number of CPU seconds of processing time consumed by the query.

Qualified Criteria: These conditions must exist for a period of time before the action is triggered. The following criteria are also “ANDed” together. If all of the criteria are met, then the exception action is triggered. Also note that the Unqualified and Qualified Criteria are “ANDed” together as well.

- **Qualification Time** – the length of time (in seconds) the condition must continue before an action is triggered
- **IO Skew** – raw number that represents the maximum difference in disk I/O counts between the average and the most busy AMP
- **IO Skew Percent** – the percentage difference in disk I/O counts between the average and the most busy AMP
- **CPU Skew** – raw number that represents the maximum difference in CPU consumption (in seconds) between the average and the most busy AMP
- **CPU Skew Percent** – the percentage difference in CPU consumption (in seconds) between the average and the most busy AMP
- **CPU millisec per IO** – the number of CPU milliseconds per disk I/O

Because skew and high CPU milliseconds per IO are situations that could occur momentarily in any legitimate query, the accumulated CPU qualification time must be specified to avoid false detections.

The exception criteria metrics are checked for using the resource usage data that has accumulated from the last exception interval until the next exception interval. The exception criterion must persist until the specified CPU qualification seconds have accumulated. The qualification time counter begins accumulating at the end of the first exception check time that detected the high CPU Milliseconds per IO.

For example, the CPU Skew has to be more than 25% for more than 600 seconds before the action is triggered.

The Unqualified and Qualified Criteria are “ANDed” together.



WD – Exception Criteria

- Exception Rules are used to detect inappropriate queries in a workload.
 - Exception Rules and Actions are specified within the WD Exception Criteria tab.
- An Exception Rule may have both **Unqualified** and **Qualified** criteria.
 - Examples of Unqualified Criteria:
 - Maximum Rows – maximum number of rows in a spool file or final result
 - IO Count – maximum number of disk I/O's performed on behalf of the query
 - CPU Time – maximum number of CPU seconds consumed by the query
 - Examples of Qualified Criteria (condition must exist for a period of time):
 - IO Skew – max difference in disk I/O Counts between the average and most busy AMP
 - CPU Skew Percent – % difference in CPU time between average and most busy AMP
- An Exception Action specifies what to do when an Exception condition is detected. Examples of actions include:
 - No exception monitoring – exception is not logged in the exception table.
 - Continue and Log – query continues and the exception is logged.
 - Change Workload – move the query into a different workload.
 - Raise Alert – no change to query; send a Teradata Manager Alert.
 - Abort – query is aborted.

TDWM – Specify Exception Criteria

After identifying the types of requests that make up the workload, you begin defining the behavior you want for those requests. Use Exception Criteria to define performance-related thresholds that trigger special handling such as aborting the requests, or continuing the requests but modifying the workload, issuing an alert, or running an external program.

Teradata checks for exception conditions at the following times.

- Synchronously – at the end of each AMP step
- Asynchronously – at the configurable time interval (1-3600 seconds); this value is set within TDWM using the left pane selection: *Settings → Intervals → Exception Interval*

Exception Actions specify what to do when an Exception condition is detected.

- No exception monitoring – exception handling is effectively turned off and exceptions are NOT logged.
- Abort – query is aborted.
- Change Workload; move the query into a different workload.
- Raise Alert; no change to query; send a Teradata Manager Alert
- Run Program; no change to query; have Teradata Manager execute a program.

Notes:

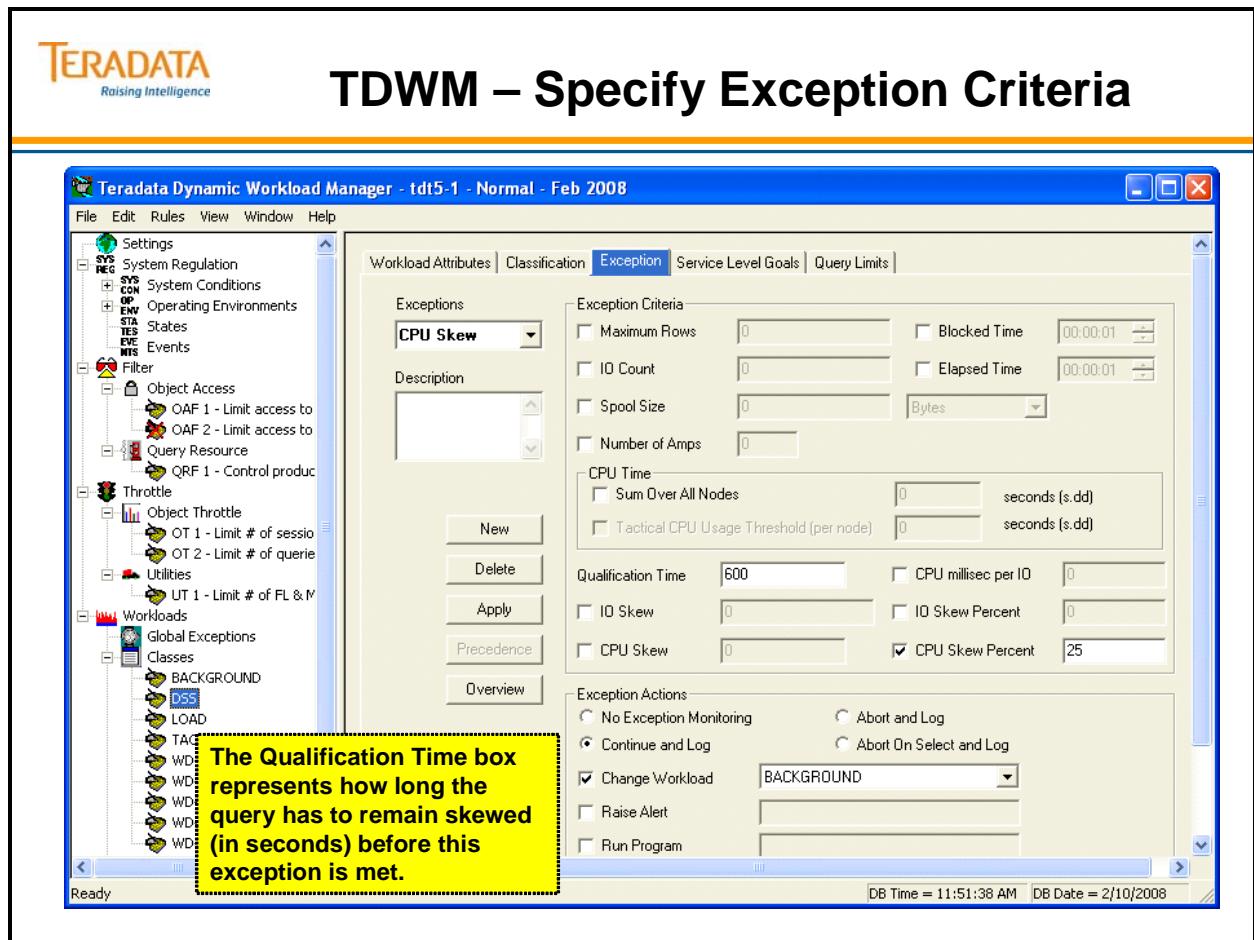
- Any exception taken on a query is automatically logged in the DBC.TDWMExceptionLog.
- Skew is NOT calculated synchronously at the end of query steps. Asynchronous exception checking is the sole method used to detect skew.

CPU Time Note

If the workload has an Enforcement Priority of tactical, initially the **Tactical CPU Usage Threshold (per node)** check box will be grayed out (not available). The **Tactical CPU Usage Threshold (per node)** check box will be enabled as soon as a positive value is specified for **Sum Over All Nodes**. This causes the Teradata Database to use PSF query milestones to move the workload to a different Allocation Group as soon as the limit is reached. Normal exception processing happens less frequently. When exception processing sees that either CPU time limit has been reached, it will perform normal exception processing.

If the workload is tactical and a positive value for **Tactical CPU Usage Threshold (per node)** is specified, then one of the Exception Actions must be **Change Workload**.

Before saving the Rule Set, TDWM checks to see that all Tactical WD's with a positive value for CPU Usage Threshold (per node) have another WD specified. The user needs to define another WD in the same Resource Partition to satisfy this condition.



Example – Exception Handling

The facing page contains an example of a screen that is accessed in Teradata Manager.

Teradata Manager → Dashboard → Virtual Utilization Summary

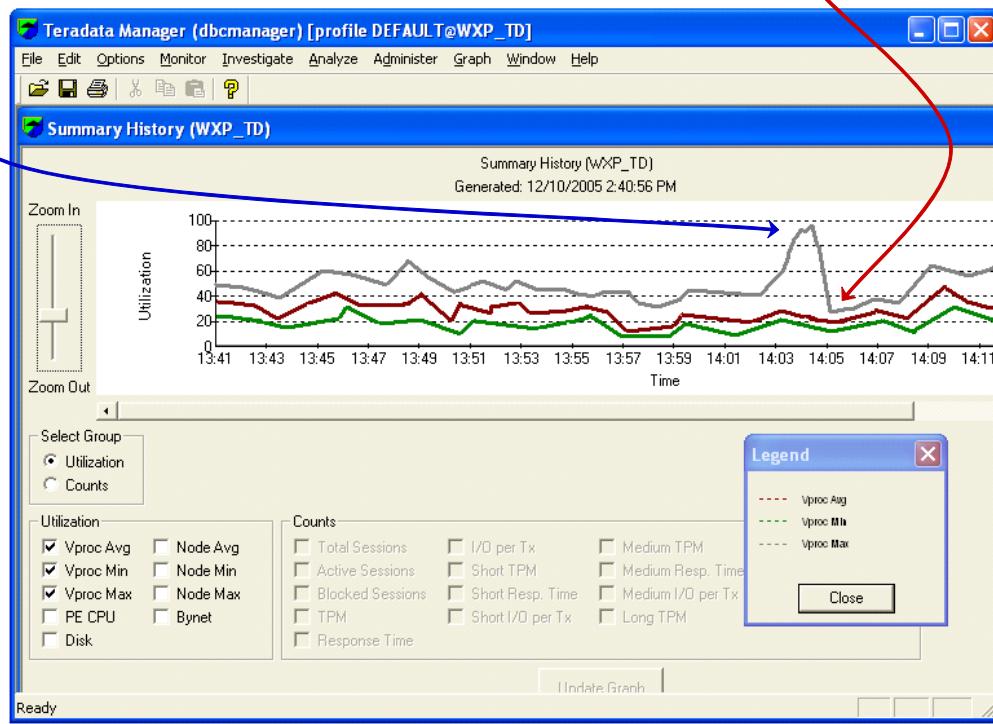
Several queries were submitted in the Strategic workload and these all caused AMP skewing. TASM automatically aborted these queries because of the exception criteria.



Example – Exception Handling

"Bad" Query causes skewing.

Query is automatically aborted by TASM.



Example – Exception Handling (cont.)

A user will receive an error message if a query is aborted because an exception criteria was exceeded.

Any exception taken on a query is automatically logged in the DBC.TDWMEExceptionLog.

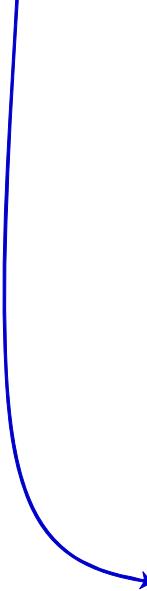
Error codes that are logged in the TDWM Exception Log are:

3149	Query request was rejected because of an Object Access Filter rule.
3150	Query request was rejected due to a Query Resource Filter rule.
3151	Query request was rejected because of an Object Throttle rule.
3152	Logon request was rejected due to an Object Access Filter rule.
3153	Logon request was rejected because of an Object Throttle rule.
3156	Request aborted by TDWM. Exception Criteria exceeded.
3158	Informational message logged by TDWM. Exception Criteria exceeded.
3162	TWM Limit for this utility type was exceeded (not returned); for load utilities.
3163	TWM Limit for all utilities has been exceeded (not returned); for load utilities.



Example – Exception Handling (cont.)

End user gets an error message describing the reason why the query was aborted.



```
Teradata SQL Assistant (WXP_TD)
File Edit View Tools Window Help
User: TFACT03 Source: WXP_TD Type: TERADATA
DBC
Tables Views Macros Function Procedur
DS TFACT TFACT01
Query
SELECT Store_num,
       Item_num,
       Saledate,
       Sumsales,
       RANK (sumsales) AS "Rank"
  FROM (SELECT store_id,
               item_id,
               sales_date,
               SUM (total_revenue)
          FROM Sales GROUP BY 1, 2, 3 )
     AS tmp (store_num, item_num, saledate, sumsales)
History
RunDate EndTime Source Elapsed Rows Result Notes SQL
12/10/2005 14:05:22 WXP_TD 00:03:44 0 SELECT Store_num, Item_num
12/10/2005 14:01:03 WXP_TD 00:00:00 25 EXPLAIN SELECT Store_num
3156: Request aborted by TDWM. Exception criteria exceeded: CPU Time, AMP CPU Skew.
Line 1 | 2:05 PM
```

Exception is logged in TASM Exception Log: DBC.TDWMExceptionLog

To view exceptions: Teradata Manager > Analyze > Trends > Workload Definition Query List

TDWM – Priority Scheduler

TDWM automatically assigns each workload definition to an Allocation Group based on the Enforcement Priority that you specified when you created the workload. You can modify the assigned Allocation Group in the **Map WD to AG** dialog box.

The Allocation Groups are weighted to determine the amount of resources allocated to workloads associated with those groups. Workloads are mapped to the same AG for all periods; however, the weighting of AGs may differ over a specific period.

The **Priority Scheduler View** is new within TDWM and provides the following capabilities:

- Use the System-Level Parameters button to set system level parameters – these are the same parameters that can also be set in PSA.
 - **System CPU Limit Percent** – limits total amount of CPU resources used by all the Teradata Database sessions across all resource partitions. Default is 100%.
 - **Reserved** – defines the number of AMP worker tasks (AWTs) reserved within each AMP vproc for work requests assigned to Allocation Groups having the Expedite attribute.
 - **Limit** – Defines the maximum number of AMP worker tasks that can run at any time. The number of tasks that run might be less than this maximum, depending on the current limit. The default is 80.
- **Map WD to AG** – modify the assigned Allocation Group for a WD.
- **Create New RP** – used to define a new Resource Partition to set up different behavior for workloads.
- **Create New AG** – used to define a new Allocation Group for your workloads.
- **Delete RP** – used to delete a Resource Partition; if AGs are associated with this RP, you need to reassign or delete them before deleting the RP.
- **Delete AG** – used to delete an Allocation Group; if WDs are associated with the AG, you need to reassign or delete them before deleting the AG.
- **Move AG to Different RP** – used to change the association between an AG and a RP.

The **Default-Weights** option (in the Rules menu) provides a version of the Compare Weights option that is similar to Priority Scheduler Administrator (PSA) in Teradata Manager. This presents a table of Resource Partitions and their associated AGs, showing their assigned and relative weights. Similar to PSA, TDWM assigns names to the AGs instead of Allocation Group IDs, making it easier to distinguish them.

TERADATA
Raising Intelligence

TDWM – Priority Scheduler

Teradata Dynamic Workload Manager - tdt5-1 - Normal - Feb 2008

File Edit Rules View Window Help

System Conditions
Operating Environments
States
Events
Filter
Object Access
OAF 1 - Limit access to
OAF 2 - Limit access to
Query Resource
QRF 1 - Control product
Throttle
Object Throttle
OT 1 - Limit # of sessions
OT 2 - Limit # of queries
Utilities
UT 1 - Limit # of FL & M
Workloads
Global Exceptions
Classes
BACKGROUND
DSS
LOAD
TACTICAL
WD-ConsoleH
WD-ConsoleL
WD-ConsoleM
WD-ConsoleR
WD-Default

State: Base
 Specify Values for this State
 System Level Parameters
 Hide Details
 Map WD to AG
 New RP
 New AG
 Delete RP
 Delete AG
 Move AG

RP Name	RP Weight	RP Rel Wgt	RP CPU Limit	AG Name	Priority	AG Weight	% Within RP	Rel Wt %	AG CPU Limit	Expedited	WD's
Tactical	60	60.0	100	Tactical	Tactical	20	100.0	60.0	100	<input type="checkbox"/>	TACTICAL
Standard	20	20.0	100	PenaltyBox	Normal	5	5.9	1.2	100		BACKGROU
				DSS	Normal	10	11.8	2.4	100		DSS
				Background	Background	10	11.8	2.4	100		WD-Conso
				Normal	Normal	20	23.5	4.7	100		WD-Conso
Default	20	20.0	100	Priority	Priority	40	47.1	9.4	100		WD-Conso
				L	Background	5	6.7	1.3	100		WD-Conso
				M	Normal	10	13.3	2.7	100		
	H	Priority	20	26.7	5.3	100					

Accept Restore

Ready DB Time = 12:05:38 PM DB Date = 2/10/2008

When Workload Definitions are activated, you must use TDWM to manage Priority Scheduler settings.

Teradata Workload Analyzer

This application provides the following capabilities:

- Migrate existing PSF settings into workload definitions.
- Establish workload definitions from query history or directly
- Can be used “iteratively” to analyze and understand how well the existing workload definitions are working, and modify those definitions if necessary.

This tool combines data from existing Priority Scheduler settings (via Priority Definition or PD sets) and workloads (via Teradata DBQL – Database Query Log) to determine workload definitions for TDWM.

This application can also apply best practice standards to workload definitions such as assistance in SLG definition and priority scheduler setting recommendations.

In addition, Workload Analyzer supports the conversion of existing Priority Scheduler Definitions (PD Sets) into new workloads. A PD set is the collection of data, including the resource partition, allocation group, period type, and other definitions that control how the Priority Scheduler manages and schedules session execution.



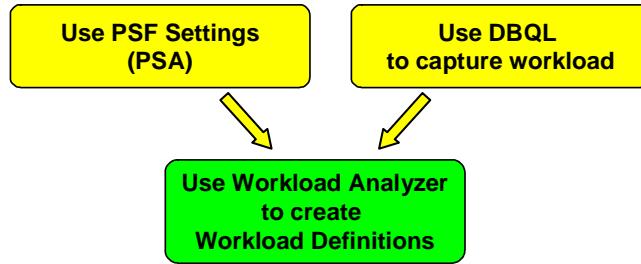
Teradata Workload Analyzer

Capabilities of Teradata Workload Analyzer include:

- Identifies classes of queries and recommends workload definitions
- Provides the ability to migrate existing PSF settings
- Recommends workload mappings and weights
- Provides recommendations for appropriate Service Level Goals (SLG)

Workload Analyzer combines data from two sources:

1. Existing Priority Scheduler settings (via Priority Definition or PD sets)
2. Workloads (via Teradata DBQL – Database Query Log)



Teradata Query Scheduler

Teradata Query Scheduler provides a Scheduled Requests capability to the Teradata Database. Scheduled Requests are user requests to submit queries for off-line execution at later or more preferable dates and times. This does not; however, guarantee that scheduled requests will be executed at the deferred time. Therefore the user must also supply a time period after the scheduled time that the request can be executed.

Teradata QS is designed to manage request inputs to your Teradata Database and keep the database running at optimum performance levels. Teradata QS consists of client and server system components, and also uses the Teradata Database called **tdwm**.

Scheduling Queries

The Teradata Query Scheduler (QS) server attempts to execute requests during the time period specified, but **restrictions still govern the execution of scheduled requests**. When end-users know of existing database rules that will prevent a SQL request from running or if they suspect their queries will overload the Teradata Database, they can actively schedule a request using the Teradata Query Scheduler Submit dialog box.

Teradata QS monitors SQL requests actively scheduled using the Teradata Query Scheduler Submit dialog box and requests scheduled because of database workload management. Teradata QS manages workloads submitted to your Teradata Database by executing the request immediately or suspending the request to run at a later time. By actively managing your Teradata Database, Teradata QS makes sure that the most important work gets done at the best time and that system resources are not overloaded.

When a request is scheduled, end-users provide information that defines preferences for when it is executed. A request can be scheduled to run periodically or only once during a specified time period without an active system user connection.

Because a scheduled request can actually be executed many times, the term **request** is used to mean the actual definition of the scheduled request parameters. The term **job** is used to mean an individual instance a scheduled request is scheduled to run. For example, a scheduled **request** can be defined to execute *daily*. That request causes a separate **job** to be created *every day* to execute that request.

When a request is submitted for scheduling by the user, the request is analyzed prior to acceptance by QS. If one or more access and/or query resource restrictions would keep the scheduled request from running over the entire time period in which it may be executed, then the request will fail. However, if there is a later time within the specified execution interval when it is not restricted, then the scheduled request will be rescheduled for some time later.

Typically, the Teradata QS client software is installed on many systems while the Teradata QS server software is installed on one or a limited number of servers. Administration of the Teradata QS capabilities is performed by a Teradata QS Administrator application.



Teradata Query Scheduler

The Teradata Query Scheduler provides a **Scheduled Requests** capability.

- Scheduled requests are SQL queries scheduled by end-users for off-line execution.
- What are the main components?
 - **TQS Server components** – software that runs on a Windows-based server that monitors the job to determine when it can be run and submits the job.
 - **TQS Client components**
 - **TQS Viewer** – to manage the request results
 - **TQS Submit Dialog Box** – submit single or multiple SQL statement requests for execution at a later date and time
 - **TQS Operations Utility** – tool to configure and start the TQS server
 - **TQS Administrator** – program to enable “scheduled requests”, set up user profiles, and setup time frames in which schedule requests are executed by TQS.
- Are there any other user applications that can also automatically schedule requests?
 - Teradata SQL Assistant does it by recognizing the DWM error codes that come from the database when a query is delayed because of a rule violation.

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Workload Management is accomplished by using several tools that assist in defining the rules that control the allocation of resources to workloads running on a system.
 - These rules include **filters**, **throttles**, and **workload definitions**.
 - Workload definitions (new with Teradata V2R6.1) are rules to control the allocation of resources to workloads.
- The benefit of TASM is to automate the allocation of resources to workloads.
- The key product that is used to create and manage these rules is Teradata Dynamic Workload Manager (TDWM).
- Other tools that facilitate in workload management include:
 - **Teradata Workload Analyzer** – helps create workload definitions by analyzing existing PSF settings and DBQL information.
 - **Teradata Manager** – contains new Workload Monitor and Trend Analysis capabilities.
 - **Teradata Query Scheduler** – facility to schedule query requests for Teradata

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. What type of TDWM filter or throttle rule is needed for the following restrictions?

Limit the number of concurrent sessions _____

Reject queries based on max processing time _____

Reject queries accessing a specific DB _____

Limit the number of FastLoad jobs _____

Delay more than 20 queries for a specific account _____

2. What is the purpose of the default workload definition name "WD-Default"?

- A. Default workload for any queries with an enforcement policy of normal.
- B. Default workload for any queries that are not associated with a workload.
- C. Default workload for any queries assigned to Default resource partition.
- D. Default workload for any queries assigned to Standard resource partition.

3. Which query attribute is not used by the Parsing Engine software to 'classify' a query into a Workload Definition (WD)?

- A. User name
- B. Account
- C. User Role
- D. User Profile
- E. Optimizer estimates

Review Questions (cont.)

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions (cont.)

4. Which Teradata application is used to activate a workload definition rule set?
 - A. Workload Analyzer
 - B. Priority Scheduler Administrator
 - C. Teradata Manager
 - D. Dynamic Workload Manager
5. Which exception action is NOT possible?
 - A. Do nothing and simply log the exception
 - B. Delay the query to off-hours
 - C. Abort the query
 - D. Send an alert to the DBA
6. Which criteria has less overhead?
 - A. Who criteria
 - B. What criteria
 - C. Where criteria
 - D. All of these have similar overhead
7. Place the following control options in the proper sequence from 1 to 4 as they are acted upon by Teradata software.
 - A. Object throttles
 - B. Exception criteria
 - C. Workload throttles
 - D. Object filters

Teradata Training

Notes

Module 51



Teradata Manager and Performance Monitor

After completing this module, you will be able to:

- List three tools of Teradata Manager.
- Use Performance Monitor to view sessions and the SQL executing within an active session.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Teradata Manager.....	51-4
Getting Started	51-6
Teradata Manager Main Window	51-8
Teradata Manager Applications	51-10
Teradata Manager Dashboard	51-12
Teradata Performance Monitor	51-14
Teradata Performance Monitor Chart Function	51-16
Teradata Performance Monitor – Resource Usage	51-18
Teradata Performance Monitor – Resource Usage Detail.....	51-20
Teradata Performance Monitor – Session Information.....	51-22
Teradata Performance Monitor – User Session Info.....	51-24
Teradata Performance Monitor – SQL & Explain Steps	51-26
Teradata Manager – Database Console.....	51-28
Database Console Example.....	51-30
Teradata Manager – Locking Logger.....	51-32
Summary	51-34
Review Questions	51-36

Teradata Manager

Teradata Manager is a production and performance monitoring system that simplifies the tasks of monitoring, controlling, and administering one or more Teradata Database systems.

As the command center for the Teradata Database, Teradata Manager supplies an extensive suite of indispensable DBA tools for managing your Teradata System. With Teradata Manager you can use a variety of specially designed tools and applications to gather, manipulate, and analyze information about each Teradata Database you need to administer.

- It provides a graphical interface that makes the tools and applications easy to use.
- You can query the Teradata Database status and utilization, present information about system performance in reports and graphs that are easy to read, administer Teradata Database users and sessions, and more.

Teradata Manager is both powerful and flexible because its structure can be customized so you can configure it to address a wide variety of issues that suit the needs of individuals or groups within your organization.

You will find it easy to run several applications simultaneously, to configure Teradata Manager to fit your specific needs, and to work with more than one instance of the Teradata Database at a time.

Teradata Manager collects, analyzes, and displays performance and database utilization information in either report or graphic format, displaying it all on Windows PC.

Teradata Manager's client/server feature replicates performance data on the server for access by any number of clients. Because data is collected once, workload on the database remains constant while the number of client applications varies. Applications that support client/server include:

Performance Monitor, Session Information, Dashboard, Trend Analysis, Resource History, Audit Reports, Alert Viewer.

Teradata Manager runs on a PC that is network-connected to the Teradata Database platform(s).

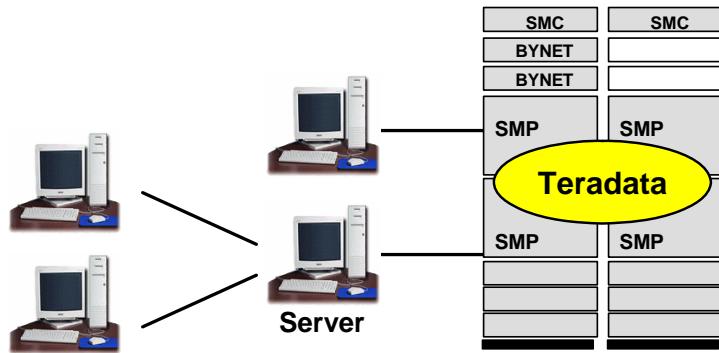


Teradata Manager

Use Teradata Manager to:

- Provides special tools and applications to gather, manipulate, and analyze information about the Teradata Database.
- Provides an easy to use Graphical User Interface.
- Can query status and utilization statistics – presents information in reports & graphs.
- Can be used to administer Teradata Database users and sessions.
- Can run several applications simultaneously.
- Runs on a network-connected PC.

In client/server mode,
applications request data
from the Teradata Manager
Service instead of logging
on to the database.



Getting Started

After the Teradata Manager application has been installed, the Teradata Database needs to be set up with the databases, tables, and macros necessary to run Teradata Manager. The RDBMS Setup application will assist you in this task.

RDBMS Setup uses your parameters to generate SQL statements for creating and modifying databases/tables/macros on the specified RDBMS. Each SQL statement and its execution status (whether it was submitted successfully or unsuccessfully) are logged in the main window of the application.

The RDBMS Setup application establishes a DBC Manager user environment and a default profile. This profile can be updated at a later time and additional profiles can be created. An example of the RDBMS Setup application is shown on the facing page.

Teradata Manager profiles can be used to control which Teradata systems can be connected to, applications that will automatically be started (and at a specific time), tailored drop-down menus, and who is allowed to modify profiles. Users can be assigned to a profile that corresponds to a user with a logon to a Teradata Database system. For security purposes, a password for the logon name is required.

With Teradata Manager 6.0, the profiles are centrally created and stored within Teradata. The Profiles tab creates profiles on the system to which you are logged on to. With previous releases of Teradata Manager, profiles were stored locally on the PC where they were created. By default, there is one profile (named DEFAULT) to which all users have access.

Some of the key fields include:

- **Database Name** - the TDPid of the Teradata Database.
- **Super User** - the name of the primary administrator for the database. This defaults to DBC.
- **Super User Password** - the string the primary administrator enters to access the database.
- **Console Password** - the console user's password. If the console user already exists, then Database Setup will log on to the database as 'console', using the password that you supply. If the console user does not exist, the 'console' user is created and assigned the supplied password. The 'console user' is used to hold "dummy" macros that represent the various remote console utilities. Permission to use a remote console utility requires execute permission on the macro.
- **Set up Database for Teradata Manager** must be checked.
- **Perm Space** - the amount of permanent space allocated to the new database.
- **Database Type** - select the type of operating system the new database is running on.
- **Spool Space** - the amount of spool space allocated to the new database. If you want the new database to inherit the same amount of space as its parent, check the **Same as Parent** option.

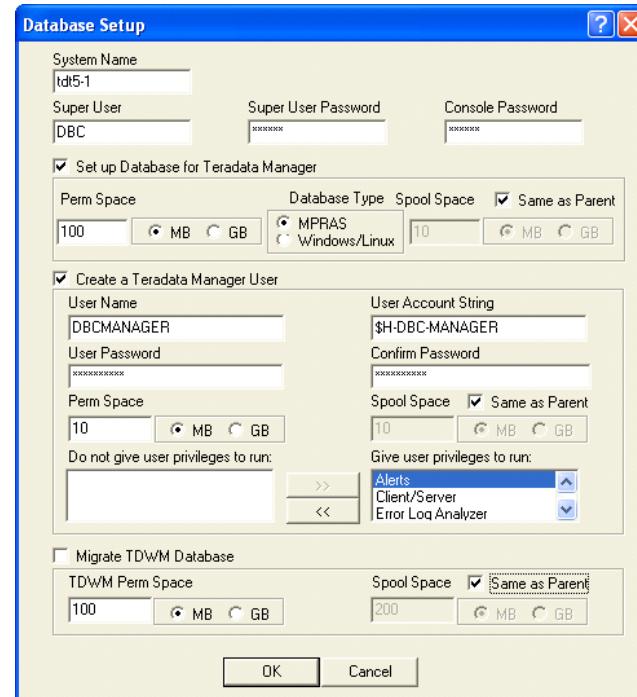


Getting Started

With Teradata Manager 12.0, initially use the RDBMS Setup application to establish a DBC Manager user environment and a default profile.

A profile corresponds to a user with a logon to a Teradata Database system and with a specific set of privileges.

- Profiles may be added/modified and are used to ...
 - establish a logon to Teradata
 - specify the applications available to a user.
 - have applications scheduled to automatically start at a specific time



Teradata Manager Main Window

The facing page contains an example of Teradata Manager's main window.

The menu layout is dynamic – depends on which applications are installed and what the profile contains. If an application is not available, it is removed from the menu - not just grayed out.

It is also possible to create (customize) your own menu which executes a set of scripts. This customization would appear in a menu choice named Local. The reference manual provides details on how to customize this menu option.

Miscellaneous Setup Notes:

The data collected by the Teradata Manager Service is used by the Trends reporting Feature. In order for the Teradata Manager Service to collect data, you need to configure, schedule and enable data collection tasks using the Data Collection tab.

Teradata Manager's Data Collection feature stores historical data in database dbcmngr.

TERADATA
Raising Intelligence

Teradata Manager Main Window

The Main Window of Teradata Manager provides access to menu functions and applications.

This example displays the Monitor menu and the output of the Node Usage function.

Teradata Manager (dbc) [profile DEFAULT@tdt1-1]

File Edit Options Monitor Investigate Analyze Administer Window Help

Dashboard Sessions Nodes Vprocs Teradata DWM Performance Monitor... Session Information...

Node Usage (tdt1-1) Generated: 4/7/2008 4:18:56 PM View as Graph

Name	Status	Uuse	Channel use	Disk Q Len	Disk reads	Disk writes	Host reads
1-04 UP	15%	0%	0%	0.0	15	51	23
1-05 UP	5%	0%	0%	0.0	28	40	0
1-06 UP	8%	0%	0%	0.0	15	43	0

Right click on a node name for additional information

View current Node performance data

Teradata Manager Applications

The facing page lists the applications that are available through the menu selections. A brief description of the key applications is provided below.

- Alert Policy Editor – define the actions that should take place when specific events occur on your Teradata machines
- Dashboard – view the overall system performance of your Teradata Database systems from a single view point
- BTEQ Window – load and extract data from the Teradata Database using a version of BTEQ with a Windows interface
- Error Log Analysis – investigate the Teradata Database error log
- Locking Logger – determine whether system performance has been degraded by an inappropriate mix of SQL statements using a table of information extracted from the transaction logs
- LogOnOff Usage – view daily, weekly, and monthly logon statistics based on information in the DBC LOGONOFF view on the Teradata Database
- Priority Scheduler Administrator – control the allocation and consumption of computer resources available to the Teradata Database
- Database (Remote) Console – run many of the Teradata Database console utilities from your Teradata Manager PC
- Session Information – get information on session status, modify session priority, view blocking/blocked sessions, change session priority
- Space Usage – monitor disk space utilization and move permanent space from one database to another.
- Statistics Collection – create, drop, and update statistics for the Teradata Database
- Teradata Performance Monitor – analyze the performance of a Teradata machine and to perform related control functions on that machine
- Historical Reports – the Analyze menu provides you with views of various historical trends in resource usage. In most cases, these trends can be viewed in either graph or table format, and can be filtered in many ways to customize your historical system view.



Teradata Manager Applications

Teradata Manager provides menu selections which access numerous functions and/or applications. The menus that provide access to these functions are:

Monitor Menu Functions

- Dashboard (requires data from server)
- Sessions (new application and display)
- Nodes
- Vprocs
- Teradata DWM (Delay Queue Stats, etc.)
- Performance Monitor
- Session Information (historical application – users may prefer this view)

Investigate Menu Functions

- Audit Log
- Error Log
- Space Usage
- Logon/Logoff History
- Locking Logger
- Alert Viewer
- System Emulation Tool
- Visual Explain

Analyze Menu Functions

- Trends
- Resource History
- Statistics Wizard
- Index Wizard

Administristrate Menu Functions

- Teradata Manager (including TM profiles)
- Teradata Database (Administrator)
- Database Console (Remote Console)
- Alert Policies
- SQL Assistant
- BTEQ Window
- Teradata Database Setup
- Statistics Collection
- Move Space
- Move Database
- Workload Management
 - Dynamic Workload Manager
 - Workload Analyzer
 - Priority Scheduler
 - Query Scheduler

Teradata Manager Dashboard

The facing page contains an example of Teradata Manager's Dashboard's top level view.

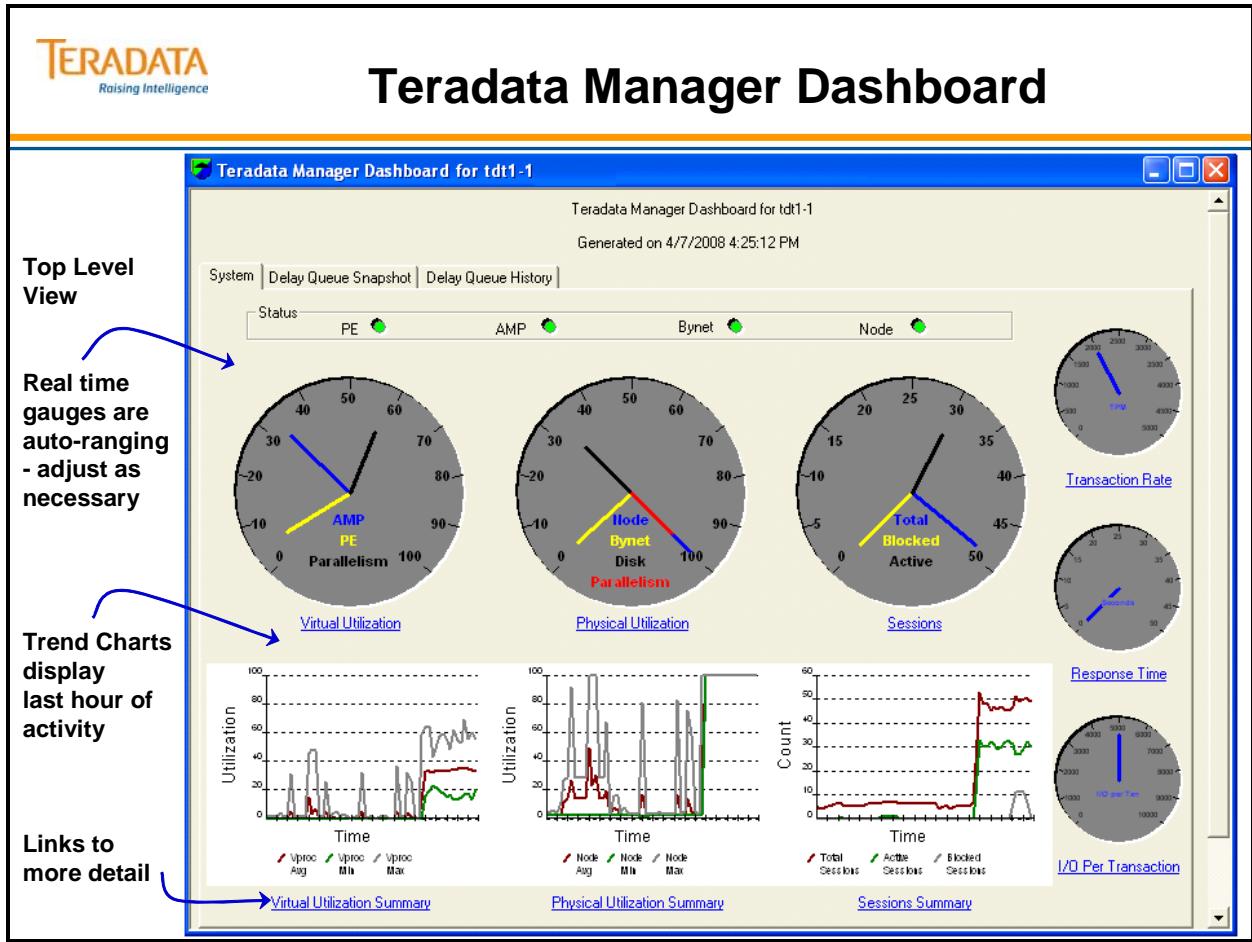
The Dashboard provides you with a summary of the current state of the system on a single page. The page contains real-time data and trend charts showing resource consumption over the last 60 minutes. You can drill down for more detailed information. Lights on the status bar at the top of the Dashboard indicate AMP, PE, Node and BYNET status: a green light indicates a status of "ok", a red light indicates a status of "down".

The top level Dashboard screen shows current and recent historical information. It permits you to see the state of the system at a glance and to predict future problems by viewing recent trends.

The benefits of this view include:

- A single screen view of current performance information and performance trends
- Ability to easily drill down to more detailed information using the "links". This can assist in rapid problem resolution.

The difference between Dashboard and Teradata Performance Monitor is that Dashboard provides a single screen image of all the information. To see session information and more detail, then use the functions of Teradata Performance Monitor.



Teradata Performance Monitor

The Teradata Performance Monitor (formerly called PMON) application may be started via Teradata Manager or may be executed as stand-alone application. Teradata Performance Monitor uses the Teradata Database PM / API to collect near real-time system configuration, resource usage, and session information from the Teradata Database either directly or through a Teradata Manager server PC. Teradata Performance Monitor then formats and displays this information as requested:

Teradata Performance Monitor allows you to analyze current performance and both current and historical session information, and to abort sessions that are causing system problems.

Teradata Performance Monitor displays this performance data on four sets of screens, one for each of the following types of data:

- General Health information (Configuration and Summary)
This data is displayed on the main screen and is refreshed automatically at a user-defined interval. This data can also be displayed as a chart of summary values over time by clicking on the Chart button.
- Resource information (Nodes and Vprocs)
This data is collected and displayed only when requested by the user. Graphs showing a user selected data point for each Node (or Vproc) may be displayed by double clicking the required data point.
- Session Information (Statistics and Lock information)
This data is collected and displayed only when requested by the user. The session summary screen shows all the sessions currently logged onto the system and allows the user to filter these sessions and to sort them in various ways. It also shows which sessions are currently blocked.

Detail session information, for a specific session may be seen by double clicking on that session. Graphs showing a user-selected data point for each Session may be displayed by double clicking the required data point. Graphs showing data point A by data point B may be displayed by clicking on one data point and dragging it to the other.

Detail lock information may be displayed by double clicking on any session that is shown to be blocking, or blocked by, another session.

You can also display session skew information and SQL Text/Explain steps.

- Historical session information
A list of historical session files is displayed. By choosing from this list the user may analyze a problem after the fact. The data itself is displayed on the regular Session Information screens.

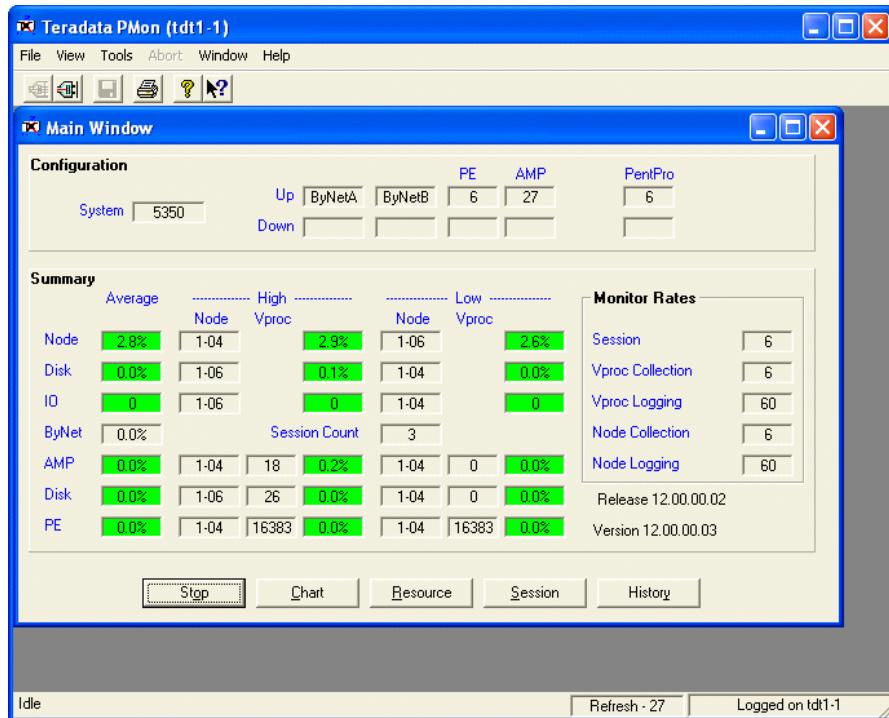


Teradata Performance Monitor

This utility can be started from Teradata Manager or executed as a stand-alone utility.

The summary display provides:

- **Highest and lowest values for AMPs, Disks, PEs, and I/Os**
- **Physical and virtual resource information**



The screenshot shows the Teradata Performance Monitor (Pmon) application window. The main window displays system configuration and performance monitoring data. Key visible data includes:

Category	Average	High	Low
Node	2.8%	1.04	2.9%
Disk	0.0%	1.06	0.1%
IO	0	1.06	0
ByNet	0.0%	Session Count: 3	
AMP	0.0%	1.04	0.2%
Disk	0.0%	1.06	26
PE	0.0%	1.04	16383

On the right side, there is a "Monitor Rates" section with values for Session (6), Vproc Collection (6), Vproc Logging (60), Node Collection (6), and Node Logging (60). Release and Version information are also shown: Release 12.00.00.02 and Version 12.00.00.03.

Teradata Performance Monitor Chart Function

To use all of the functions in Teradata Performance Monitor, you must have the following user privileges on the Teradata Database using the GRANT MONITOR command.

- SETSESSRATE
- SETRESRATE
- MONSESSION
- MONRESOURCE
- ABORTSESSION

Teradata Performance Monitor – Chart function

The Performance Chart reflects average resource usage over time.

Select View, then Chart from the menu bar, or click the *Chart* button. The Performance Chart dialog is displayed.

Select the data points you want charted. The chart will update automatically at the defined refresh rate. (The refresh rate is defined by selecting Tools, then Options and selecting the Start tab.)

To change the time period displayed on the chart, increase or decrease the number in the Intervals field. The total time covered by the chart is the number of intervals multiplied by the interval duration. The interval duration is the Refresh Rate specified on the Start tab of the Options screen.

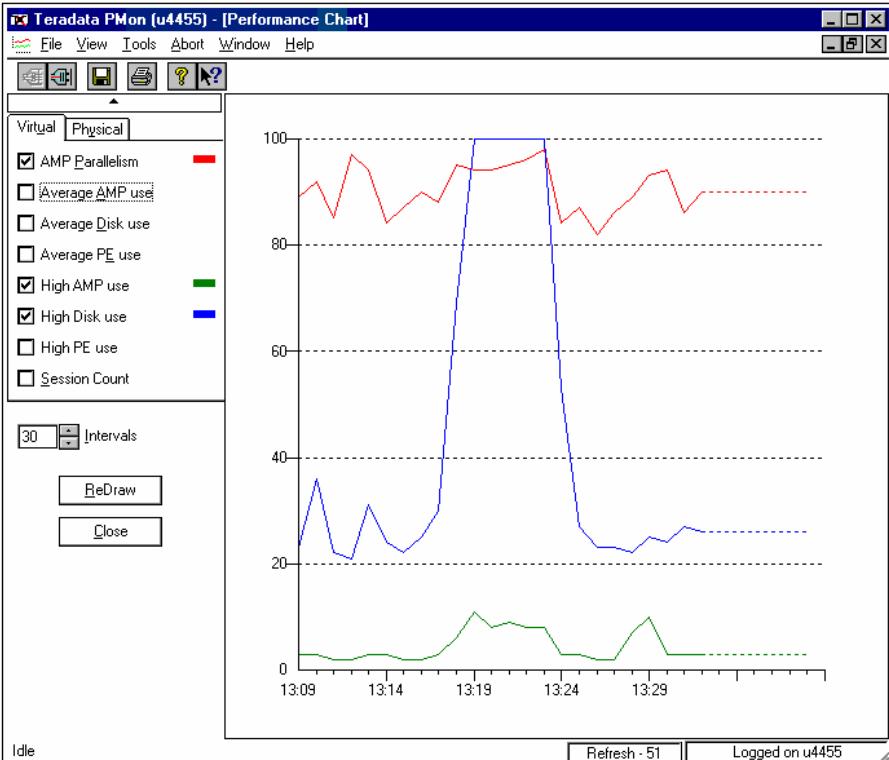
To update the chart with the new data points instantly (before the specified update interval), click on the Redraw button.



Teradata Performance Monitor Chart Function

Chart function:

- This screen allows you to display a chart of the Resource Summary values over time.
- The check boxes allows you to choose which values will be charted.
- You can chart both Physical data (Node) and Virtual data (AMP & PE) at the same time.



The screenshot shows the Teradata Performance Monitor (PMon) interface. The title bar reads "Teradata PMon (u4455) - [Performance Chart]". The menu bar includes File, View, Tools, Abort, Window, and Help. The toolbar has icons for Refresh, Stop, and Close. A legend on the left indicates that red represents "AMP Parallelism" and blue represents "High Disk use". Below the legend, there's a dropdown for "Intervals" set to 30, and buttons for "ReDraw" and "Close". The main area displays two lines: a red line for AMP Parallelism and a blue line for High Disk use. The x-axis shows time from 13:09 to 13:29. The y-axis ranges from 0 to 100. Both lines fluctuate, with the blue line showing a significant spike reaching nearly 100 around 13:19.

Teradata Performance Monitor – Resource Usage

The Resource Usage screen shows the status of the BYNETs and processors for the most recent sampling period.

Processor status displays on the left side of the screen. Processors are listed by type and processor id in either the Online list box or the Offline list box.

Resource usage statistics are displayed in the panel on the right side of the screen. A graphical representation of specific types of resource information may be viewed by double-clicking on the field that interests you. The value of the selected field will be displayed for each processor.

Graphs can be produced from statistics gathered for the User Information or Resource Information screens.

TERADATA
Raising Intelligence

Teradata Performance Monitor Resource Usage

Resource Usage:

- The Resource Usage screen shows the status of the BYNETs and processors for the most recent sampling period.
- Resource usage statistics are displayed on the right side of the screen.
- By double-clicking on the field that interests you, a graphical representation of the selected field will be displayed in a separate screen.

Vproc Type	AMP	Disk Reads	0
Status	Up	Disk Writes	830
Node Id	1-01	Avg Disk Req	1
Vproc Id	0	Host Block Reads	
Host Id / Cluster	0	Host Block Writes	
Session Log Count		Mem Segs Allocated	1,047
Session Run Count	0	Mem KB Allocated	364,756
CPU Use	7.4%	Net Reads	451
PDE User Service	0.9%	Net Writes	57
AMP Worker Task	6.6%	Backup Segs Alloc	0
PE Parser			
PE Dispatcher			
AMP Disk Use	100.0%		
CIC Use			

Interval: 60 Average AMP use: 8% Parallel Efficiency: 92%

Idle Logged on u4455

Teradata Performance Monitor – Resource Usage Detail

Graphs can be produced from statistics gathered for the User Information or Resource Information screens.

The check boxes allow you to choose the type of values that will be graphed. These buttons will only be displayed when you are graphing something of the form Variable A by Variable B.

The View menu allows you to choose what type of graph (Line graph, Pie Chart, etc) you wish to display. You should note however that Pie graphs can not be selected if you request that more than one value type be graphed at the same time. Once the graph is displayed, you can use the Select Color option to change the color of plotted items.

The Style menu allows you to select the Graph Style. The available styles (Horizontal, Vertical, Z-Cluster, etc.) will depend on both the Graph Type you have chosen and the number of value types you are plotting.

The File menu allows you to save the graph as a bitmap using the Save As command. Alternatively you may save the data that was used to create the graph to a Tab delimited file.

If the graph is too wide to fit entirely on one screen a scroll bar will be displayed at the bottom of the screen to allow you to see other parts of the graph.

TERADATA
Raising Intelligence

Teradata Performance Monitor Resource Usage Detail

Resource Usage Detail:

- This view of a Resource Usage display provides options to display details on various resources.
- This example shows AMP Worker tasks per CPU.
- The View menu allows you to choose what type of graph (Line graph, Pie Chart, etc.) you wish to display.

The screenshot shows a bar chart titled "Teradata PMon (u4455) - [AMP Worker Task by Processor - As Of - 6/20/2003 2:13:14 PM]". The chart displays the number of AMP Worker tasks per processor (AMP 0 to AMP 7). The Y-axis represents the count of tasks, ranging from 0.0 to 0.8. The X-axis lists the processors. The data is as follows:

Processor	Tasks
AMP 0	~0.30
AMP 1	~0.30
AMP 2	~0.68
AMP 3	~0.52
AMP 4	~0.32
AMP 5	~0.28
AMP 6	~0.36
AMP 7	~0.38

Idle

Logged on u4455

Teradata Performance Monitor – Session Information

This screen provides access to the detail session and lock data.

This display is divided into 5 screen partitions. There are five list boxes, an interval box, six buttons and a status bar on the screen. The list boxes are as follows:

- Filter By
- Sort by/Display
- Users
- Blocked
- Blocking

The buttons are as follows:

- Lock Info* – displays the Lock Information screen for the highlighted user
- User Info* – displays the User Information screen for the highlighted user
- Abort – aborts the highlighted user session(s)
- All Locks – Re-displays all Blocked and Blocking users
- Refresh – refreshes all data on this screen
- Close – closes this screen

* These buttons display screens that are considered to be children of the Session Screen.

When you close this screen, the data may be saved in a History file for later analysis. This feature is controlled by a setting on the Session tab of the Tools, Options screen. History files may be requested in order to help with problem resolution.

You may choose to abort a session from this screen.

1. Select Tools > Options from the menu bar, then select the Session tab.
2. Make sure the Enable Abort Session Buttons option is selected, then click OK.
3. Go to the Session dialog by clicking the Session button, or by selecting View > Session from the menu bar.
4. Select the desired session in the Users, Blocked, or Blocking windows, then click the Abort button.
5. Click Yes on the Abort Confirmation dialog to abort the session.

Note: The Abort button is also available on the User Info dialog.

TERADATA
Raising Intelligence

Session Information

- Displays a list of sessions and may be sorted by different criteria.

Options include:

- User Info – displays the User Information screen for the selected user
- Lock Info – displays the Lock Information screen for the selected user.
- Abort – aborts the highlighted user session(s).
- All Locks – Re-displays all Blocked and Blocking users.

Teradata Performance Monitor Session Information

The screenshot shows the Teradata PMon interface with the title "Teradata PMon (W2000_TD) - [Sessions - W2000_TD - 12/13/2003 4:47:48 PM]". The main window displays session information for three users: TFACT01, TFACT03, and DBC. The CPU usage for TFACT01 is 100.25%, for TFACT03 is 96.14%, and for DBC is 0.04%. The window includes filters for PE, Host, and AMP, and sorting options for Name, Total AMP CPU, Request CPU, etc. Buttons for Refresh, Close, User Info, Abort, Lock Info, and All Locks are visible at the bottom. A note at the bottom right says "Logged on W2000_TD".

User	CPU Usage (%)
TFACT01	100.25
TFACT03	96.14
DBC	0.04

To abort sessions:

1. Enable Abort Session Buttons (Tools – Options – Session tab)
2. User requires the ABORTSESSION access right.

Teradata Performance Monitor – User Session Info

This screen displays information about the selected User Session. It is selected from:

- Sessions dialog from either the Users, Blocked or Blocking list box, or
- Lock Info dialog from the Blocked or Blocking list box.

The Backward and Forward buttons allow you to scroll through the sessions that were displayed in the Users, Blocked or Blocking list box without having to manually jump back and forth between the User Info screen and the Sessions screen or the Lock Info screen.

The Modify button allows you to change session account/priority strings for active sessions.

Graphical View – you may view specific types of user information graphically by double-clicking on the field that interests you. In this case a count, or a total, will be displayed for each session, depending on which value makes sense for that type of field.

From this screen, you can also modify the priority of a session or abort a session.

Modify a session account/priority string

1. Press the Session button on Teradata Performance Monitor's main window, to display the Session screen.
2. Select the session you wish to modify in the Users list box, then press the User Info button to display the User Information screen.
3. Press the Modify button to display the Modify Session dialog.
4. The original account string will be displayed in the data entry field. Replace this with the new account/priority string you wish to use.
5. Select Apply to Current Request Only if you want this change to apply only to the currently active request, or select Apply to All Requests if you want to change the session account/priority string from this point on.
6. Select OK to execute the command and close the dialog.

Notes:

- In order to modify a session account/priority string, the logon user will require the ABORTSESSION privilege.
- Performance Monitor will not show the current priority if you modified the account for only the currently running request.

You can also abort a session from this display.

1. Select Tools > Options from the menu bar, then select the Session tab.
2. Make sure the Enable Abort Session Buttons option is selected, and then click OK.
3. Go to the Session dialog by clicking the Session button, or by selecting View > Session from the menu bar.
4. Click Yes on the Abort Confirmation dialog to abort the session.



Teradata Performance Monitor User Session Info

User Session Information

- Displays details about a specific session

Options include:

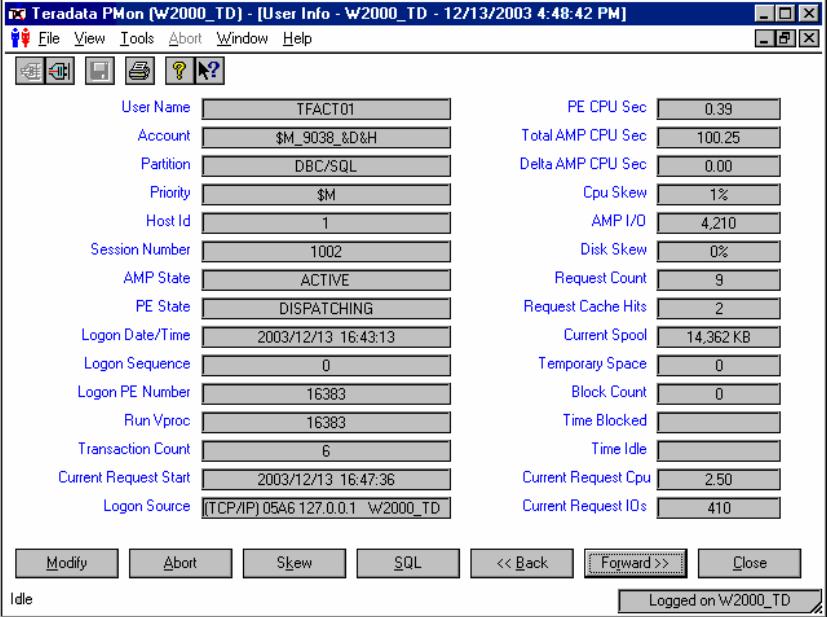
- **Modify** – change the account string and priority of the session.
- **Abort** – aborts the user session.

Abort Confirmation

Host: 1
Session No: 1002
User Name: TFACT01

Additional Options include:

- **Skew** – displays which AMPs have lowest and highest CPU and I/O values.
- **SQL** – displays the SQL text and the Explain text and the current step is highlighted.
- **Forward/Back** – move between sessions without going back to previous screen.



The screenshot shows the Teradata PMon User Session Info window. It displays session details like User Name (TFACT01), Account (\$M_903B_&D&H), Partition (DBC/SQL), Priority (\$M), Host Id (1), Session Number (1002), AMP State (ACTIVE), PE State (DISPATCHING), Logon Date/Time (2003/12/13 16:43:13), Logon Sequence (0), Logon PE Number (16383), Run Vproc (16383), Transaction Count (6), Current Request Start (2003/12/13 16:47:36), and Logon Source ((TCP/IP) 05A6 127.0.0.1 W2000_TD). On the right, it shows performance metrics such as PE CPU Sec (0.39), Total AMP CPU Sec (100.25), Delta AMP CPU Sec (0.00), Cpu Skew (1%), AMP I/O (4,210), Disk Skew (0%), Request Count (9), Request Cache Hits (2), Current Spool (14,362 KB), Temporary Space (0), Block Count (0), Time Blocked (0), Time Idle (0), Current Request Cpu (2.50), and Current Request IOs (410). At the bottom are buttons for Modify, Abort, Skew, SQL, << Back, Forward >>, and Close, along with a status bar that says "Logged on W2000_TD".

Teradata Performance Monitor – SQL & Explain Steps

The facing page contains an example of using the SQL button to display the SQL and the Explain steps for a currently running query.

TERADATA
Raising Intelligence

Teradata Performance Monitor SQL & Explain Steps

SQL Button

- This button can be used to display the currently executing SQL statement and the associated EXPLAIN text.
- The current step is also highlighted.

The screenshot shows two windows of the Teradata Performance Monitor. The top window is titled 'Current Request for session 1002' and displays the SQL statement: 'UPDATE new_sales SET store_id = store_id + 1000000;'. The bottom window is also titled 'Current Request for session 1002' and shows the execution plan with 6 steps. Step 4 is highlighted in yellow and has a blue border, indicating it is the current step. The table data is as follows:

Step	Est. Time	Actual Time	Est. IOs	Actual IOs	Description
1	0:00.00	0:00.00	0	1	First, lock TFACT."pseudo table" for write
2	0:00.00	0:00.00	0	2	Next, we lock TFACT.new_sales for write.
3	0:10.12	0:38.13	485890	484450	We do an All-AMPs RETRIEVE step from 1 which is redistributed by hash code to all AMPs
4	0:00.00		0		We read table ids out of table new_sales and redistribute them to all AMPs
5	0:00.00		0		We do a MERGE into table new_sales from new_sales
6	0:00.00		0		We send out an END TRANSACTION step

Teradata Manager – Database Console

The Database Console application allows you to run many of the Teradata Database console utilities from your Teradata Manager PC.

With the Database Console utility, you can also:

- Redirect output
- Create an autostart script
- Import or export ANSI scripts

The script feature of Database Console allows you to record commands associated with starting, running, and stopping console utilities. The commands you enter are saved as scripts and can be played back at a later time. Scripts may be run interactively from the main menu via the Scripts, then Run command, or in batch mode by setting up an Auto start entry in the Executive.

When you select a utility to run, the output for the utility appears on the main window. The input and the output for all console utilities are stored in the same Database Console log.

Console activity requires special access rights, BUT does not require UNIX Root authority.

TERADATA
Raising Intelligence

Teradata Manager Database Console

The Database Console application allows you to execute console utilities from Teradata Manager:

- **Abort Host**
- **Check Table**
- **Configure**
- **DBS Control**
- **Ferret**
- **Gateway Global**
- **Lock Display**
- **Operator Console**
- **Priority Scheduler**
- **Query Configuration**
- **Query Session**
- **Recovery Manager**
- **Show Locks**
- **Vproc Manager**
- **TDWM Dump**

The screenshot shows the Teradata Manager Database Console interface. The title bar reads "Teradata Manager Database Console". Below it, a sub-header says "The Database Console application allows you to execute console utilities from Teradata Manager:". A list of utilities is provided, with "DBS Control" highlighted. The main pane is titled "Remote Console (System tdt1-1)" and shows the "DBS Control" menu. The "General Fields" section displays several parameters with their values:

Parameter	Value
Session ID	= 7
Session Date	= TRUE (2007-12-18 12:34)
Session Duration	= 240
Session Type	= 5 (Universal)
Session Mode	= 0 (Teradata)
Lock Logger	= TRUE
Rollback Priority	= FALSE
Max Load Tasks	= 5
Roll Forward Lock	= FALSE
Max Decimal	= 15
Century Break	= 0
Date Form	= 0 (IntegerDate)
System Time Zone Hour	= 0

At the bottom right of the console window, there is a "READY" indicator.

Database Console Example

The example on the facing page illustrates the use of the Ferret > Showspace utility.

Supported utilities include:

- Abort Host (ABORT HOST) – cancel all outstanding transactions on a host that is no longer operating.
- Check Table (CHECKTABLE) – check for inconsistencies in internal data structures and for corruption.
- Configure (CONFIG) – review and update the configuration of the Teradata Database.
- DBS Control (DBSCONTROL) – used to set/display system-wide parameters.
- Ferret (FERRET) – display and set various disk space utilization attributes, and dynamically reconfigure the data on the disks to correspond with the selections.
- Gateway Global (GTWGLOBAL) – inspect and modify the gateway's operating parameters for any network attached to the associated Teradata Database.
- Lock Display (LOKDISP) – displays locks (other than HUT locks) currently present in system
- Operator Console (CNSCONS) – run supervisor commands to manage the programs that perform Teradata Database operations.
- Query Configuration (QRYCONFIG) – display the current Teradata Database configuration.
- Query Session (QRYSESSN) – monitor the state of database sessions on logical host IDs attached to the Teradata Database.
- Recovery Manager (RCVMANAGER) – monitor the progress of a Teradata Database recovery.
- Show Locks (SHOWLOCKS) – provide information about host utility locks
- Vproc Manager (VPROCMANAGER) – used to manage virtual processors



Teradata Manager Database Console Example

The Database Console capability runs many of the Teradata Database console utilities from your Teradata Manager PC.

This example shows execution of the Ferret > Showspace utility.

```

Remote Console (System u4455)
Utilities Scripts Edit View Options Help
[?] [?]

SHOWSPACE results for the Whole AMP

+-----+-----+-----+-----+-----+
| Perm Data Cyls | Spool Cyls | Temp Cyls | Jrnal Cyls | Bad Cyls |
+-----+-----+-----+-----+-----+
| Avg % of      | Avg % of      | Avg % of      | Avg % of      | % of      |
| total util total util total util total util total util total |
| Avail per Avail per Avail per Avail per Avail per Avail per Avail |
Num DSU   Cyls|Cyl Cyls #Cyl|Cyl Cyls #Cyl|Cyl Cyls #Cyl|Cyl Cyls #Cyl|Cyls |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
0    9163 | 78% 4% 397| 14% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1    9163 | 80% 5% 454| 14% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2    9163 | 75% 4% 389| 14% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3    9163 | 75% 4% 367| 14% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4    9163 | 77% 4% 336| 14% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5    9163 | 78% 4% 328| 16% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
6    9163 | 75% 4% 362| 16% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
7    9163 | 75% 4% 385| 14% 0% 3| 0% 0% 1| 0% 0% 1| 0%
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
TOTAL 73304 |           3018|          24|          8|          8|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

showspace /s

For Help, press F1

BUSY NUM

Teradata Manager – Locking Logger

The Locking Logger, sometimes called the Dumplocklog, is an optional Teradata RDBMS feature that maintains ongoing logs of transaction, session and lock object identifiers, as well as the lock levels associated with executing SQL statements that have been delayed because of database lock contention. This utility can be used to determine whether system performance has been degraded by an inappropriate mix of SQL requests.

The Locking Logger console utility creates a table of information extracted from the transaction logs. Use this table to determine if system performance has been degraded by an inappropriate mix of SQL statements.

The lock log buffers hold approximately 810 entries in a circular fashion—when the buffer is full, the next entry overwrites the first (providing a continuously updated log of the last 810 transaction locks encountered by each database AMP).

Retrieving Information

The lock log buffers are internal system-level tables. As such, you cannot access them directly. To retrieve information from the lock log buffers, you must first use the Locking Logger console utility to create a standard Teradata SQL lock log table. Then, use a series of SQL queries to join the lock log table with the system DBase, TVM, and EventLog tables. These steps convert the system-level object and session identifiers into actual database, table, user, and host names for your reports.

Enabling the Locking Logger Utility

To use the Locking Logger, you must enable it by setting the Locking Logger flag in the DBS Control utility. You must enable the Locking Logger option before you can use the Locking Logger console utility.

For information about the DBS Control utility, refer to the DBS Control chapter in *Teradata RDBMS Administrator Utilities*.



Teradata Manager – Locking Logger

Locking Logger

- Use this report to determine if an inappropriate mix of SQL requests impacts performance.
- DBS Control parameter #9 (LockLogger) must be set to True.

Report opened on Thursday, April 14, 2005 at 2:23:44 PM

ReqDate	ReqTime	Delay	Table Name
2005/04/14	07:54:12	00:00:14.000	DBC
			AccLogRuleTbl
			BlockedLevel
			BlockedMode
			BlockedHostID
			RowHash
			Rd
			1
			Blocked User Name
			\$M_ED&H
			BlockedLogonSource
			(TCP/IP) 0405 149.25.74.202
			BlockingLevel
			BlockingMode
			BlockingHostID
			Table
			Wr
			1
			Blocking User Name
			Blocking Account Name
			TLJC04
			\$M_ED&H
			BlockingLogonSource
			(TCP/IP) 04E2 149.25.74.252
			Processor
			DeadLock
			MultiBlockers
			Statement
			0-5 N N Select
ReqDate	ReqTime	Delay	Table Name
2005/04/14	07:54:33	00:00:02.000	DBC
			AccLogRuleTbl
			BlockedLevel
			BlockedMode
			BlockedHostID
			RowHash
			Rd
			1
			Blocked User Name
			TLJC06
			\$M ED&H

READY

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Teradata Manager features a variety of tools and applications that gather and analyze information about your Teradata Database system.
- Teradata Manager tools include Teradata Performance Monitor, Remote Database Console, Locking Logger, etc.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. List three tools of Teradata Manager.

Teradata Training

Notes

Module 52



Performance Monitoring

After completing this module, you will be able to:

- Identify the purpose of various ResUsage tables.
- List two ways in which ResUsage data can be collected.
- Name the two phases of gathering resource usage data and briefly describe them.
- Describe the purpose of the Teradata System Emulation Tool

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Performance Monitoring Tools	52-4
SAR and xperfstate	52-4
Why Collect Resource Usage Data?	52-6
Resource Usage Data	52-8
Data Collection	52-8
Data Logging.....	52-8
Collection Costs	52-8
Cost	52-8
Filling the ResUsage Tables.....	52-10
Collection and Log Buffers.....	52-10
Real-time Performance Monitoring	52-10
Resource Usage Logging	52-10
Specifying ResUsage Tables and Logging Rates.....	52-12
Specifying Tables using the xctl or ctl Utility	52-12
Specifying Tables using Supervisor Commands.....	52-12
Example using MultiTool	52-12
Resource Usage Tables	52-14
Setting Resource Logging from DBW	52-14
Setting Resource Logging from the Control Utility – ctl or xctl	52-14
Resource Usage Views.....	52-16
ResGeneralInfoView	52-16
ResCPUUsageByAMPView	52-16
ResCPUUsageByPEView	52-16
ResShstGroupView	52-16
ResSldvGroupView	52-16
ResSvdskGroupView (Teradata 12.0)	52-16
Resource Usage Macros	52-18
Example Output from DBC.ResNode Macro	52-20
PM/API and Performance Monitor	52-22
How PM/API Collects Data	52-22
Collecting and Reporting Resource Usage Data	52-22
Collecting and Reporting Session-level Usage Data	52-22
Example	52-22
Teradata System Emulation Tool (Teradata SET)	52-24
Performance Monitoring Summary	52-26
Review Questions	52-28

Performance Monitoring Tools

The facing page identifies the performance monitoring tools and the platforms on which they reside.

The Teradata Database provides several facilities you can use to monitor database performance.

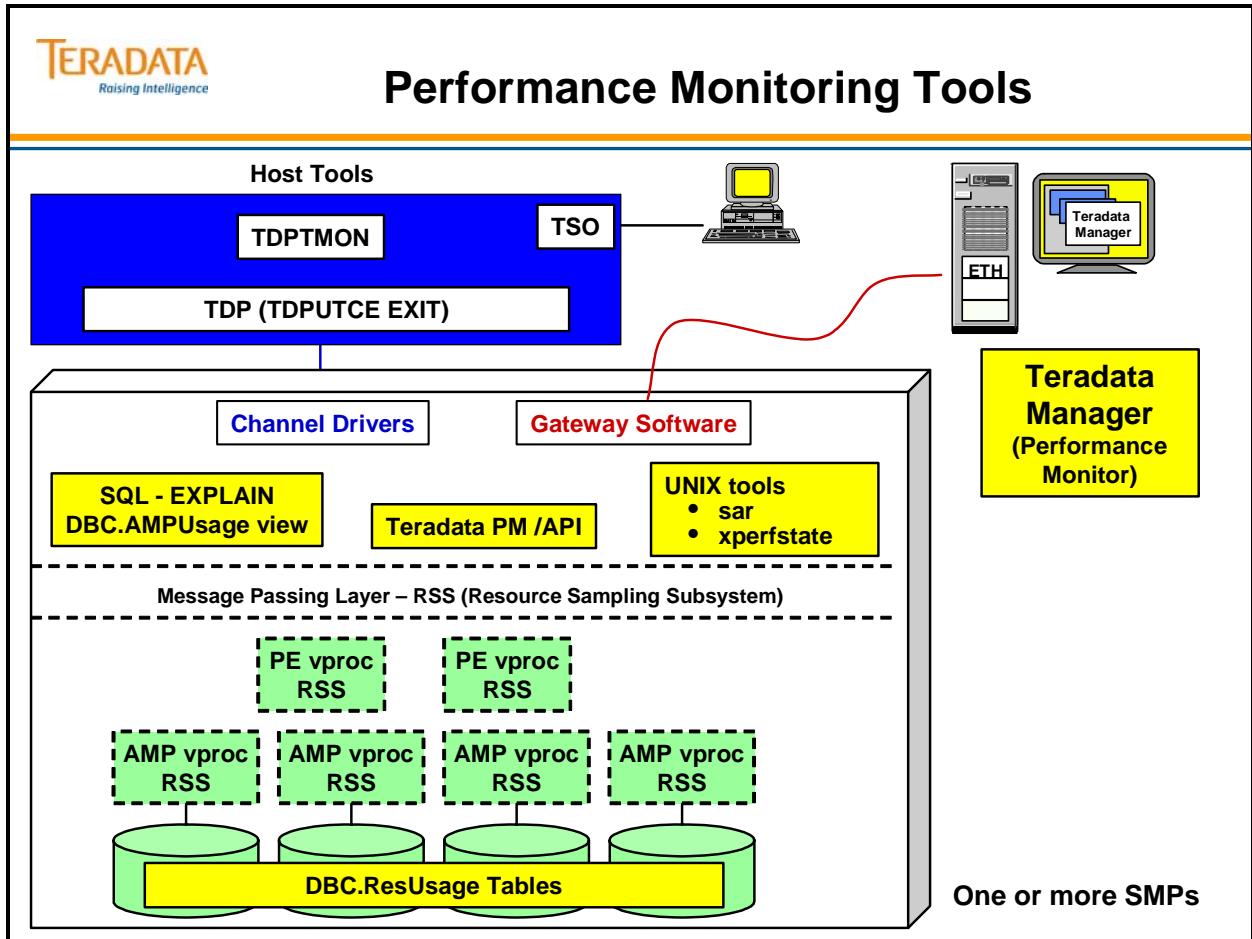
- EXPLAIN statement
- Access logging – record the activity of specific users.
- AMPUsage – a view that provide AMP Usage information for each user and account.
- ResUsage – a set of tables, view, and macros used to record and monitor system resource usage.
- Teradata PM/API – Performance Monitor / Application Programming Interface
- Teradata Manager – Performance Monitor (formerly PMON)
- Database Console Utilities – (e.g., Showspace displays space utilization)
- UNIX tools (SAR and xperfstate)
- TDP User Transaction Monitor (TDPTMON)

For example, you may have to use several of these tools to monitor query performance. You should always have an EXPLAIN report to understand what the query is doing.

SAR and xperfstate

SAR (System Activity Reports) is a UNIX facility to capture performance metrics at the UNIX operating system level.

Xperfstate is a UNIX tool that provides a real-time display of system performance combining some UNIX and Teradata information.



Why Collect Resource Usage Data?

Traditionally, resource usage (ResUsage) data and reports have been the primary diagnostic tool available to analyze performance data on a Teradata system.

The facing page shows some uses for ResUsage reports and data.

When considering system expansion and doing capacity planning, some of the activities to consider include:

- Batch windows – how much time is available to perform batch SQL jobs. A larger system with more AMPs usually means that these jobs can run faster. However, if the network and/or channel are the bottleneck, then maybe additional network and/or channel connections may be necessary.
- Backup windows – how much time is available to perform Archive activities? If a disaster occurred and a RESTORE/RECOVERY had to be performed, how long can the system be unavailable before there is negative impact to the business?
- Maintenance Windows - how much time is available to perform batch load/unload activities? A larger system with more AMPs usually means that the load/unload functions will take less time. However, if the network and/or channel is the bottleneck, then maybe additional network and/or channel connections may be necessary.
- Ad-hoc decision support queries – how much spool is needed? When will these be executed



Why Collect Resource Usage Data?

Resource Usage Data may be used to:

- Measure system benchmarks.
- Measure component performance.
- Analyze performance degradation and improvement.
- Identify potential performance impact.
- Identify bottlenecks, parallel inefficiencies and other problems.
- Assist on-site job scheduling.
- Plan installations.
- Capacity planning – resource usage data can help determine if system expansion is necessary.

Resource Usage Data

ResUsage data gathering is a two-phase process that encompasses data collection and data logging. The ResUsage facility consists of a set of tables, views, and macros to access system metrics.

Two Teradata subsystems work in conjunction with other subsystems to gather ResUsage data:

- Parallel Database Extension (PDE)
- Resource Sampling Subsystem (RSS)

Data Collection

During the data collection phase, both PDE and RSS gather information from the operating system and from Teradata Database. This data is temporarily stored in shared data collection buffers. Data collection continues for a user-specified period of time called the collect interval.

Data Logging

In the logging phase, RSS writes all gathered data to ResUsage tables and reinitializes the shared data collection buffers for the next log interval.

Collection Costs

Recording information in the DBC.ResUsage table requires disk space and processing time. Despite the additional resources used in performance monitoring, there are benefits to understanding how your system resources are being used.

Cost

The collection of ResUsage data incurs associated system overhead costs in three areas: I/O capacity, User DBC Perm Space and CPU utilization. The CPU has to write new rows to the ResUsage table depending on the preset logging interval. This increases CPU utilization during the collection process. In addition, the new rows added to the ResUsage tables require more perm space to hold the added data in user DBC where the table resides.

The costs for collecting ResUsage data depend on the table-logging interval, the number of active tables, and on the physical and virtual configuration of your system.



Resource Usage Data

The system gathers ResUsage data in a two-phase process:

- **Data collection**
- **Data logging**

Two Teradata subsystems work in conjunction with other subsystems to gather ResUsage data:

- Parallel Database Extension (PDE)
- Resource Sampling Subsystem (RSS)

Data collection

- PDE and RSS help to collect data and gather information from the operating system and the Teradata Database software.

Data logging

- RSS writes the collected data to ResUsage tables.

Collections Costs

- **Disk Space**
- **Additional I/Os (minimal)**
- **Additional CPU overhead (minimal)**

Filling the ResUsage Tables

ResUsage information is gathered in three ways, depending on the nature of the data being collected.

- Counted – the number of times an event happened. The “gather or live” buffer is updated at each event.
- Time monitored – determines how much time was spent in a particular state. The “gather or live” buffer is updated at each state change.
- Tracked data – uses a snap shot of a queue length at the collect period. This information goes directly into the collect buffer.

Collection and Log Buffers

You can collect data at the same or smaller interval than at which data is logged.

Statistical information is stored in the gather buffer and holds it there until the set collect interval. At that time, the utility moves the data to the collect buffer.

Data then moves from the collect buffer to the work buffer, where it is held until the set log interval is reached. When the set log interval is reached, RSSMon moves the data to the log buffer.

The data in the log buffer is written to ResUsage tables in the Teradata database.

Real-time Performance Monitoring

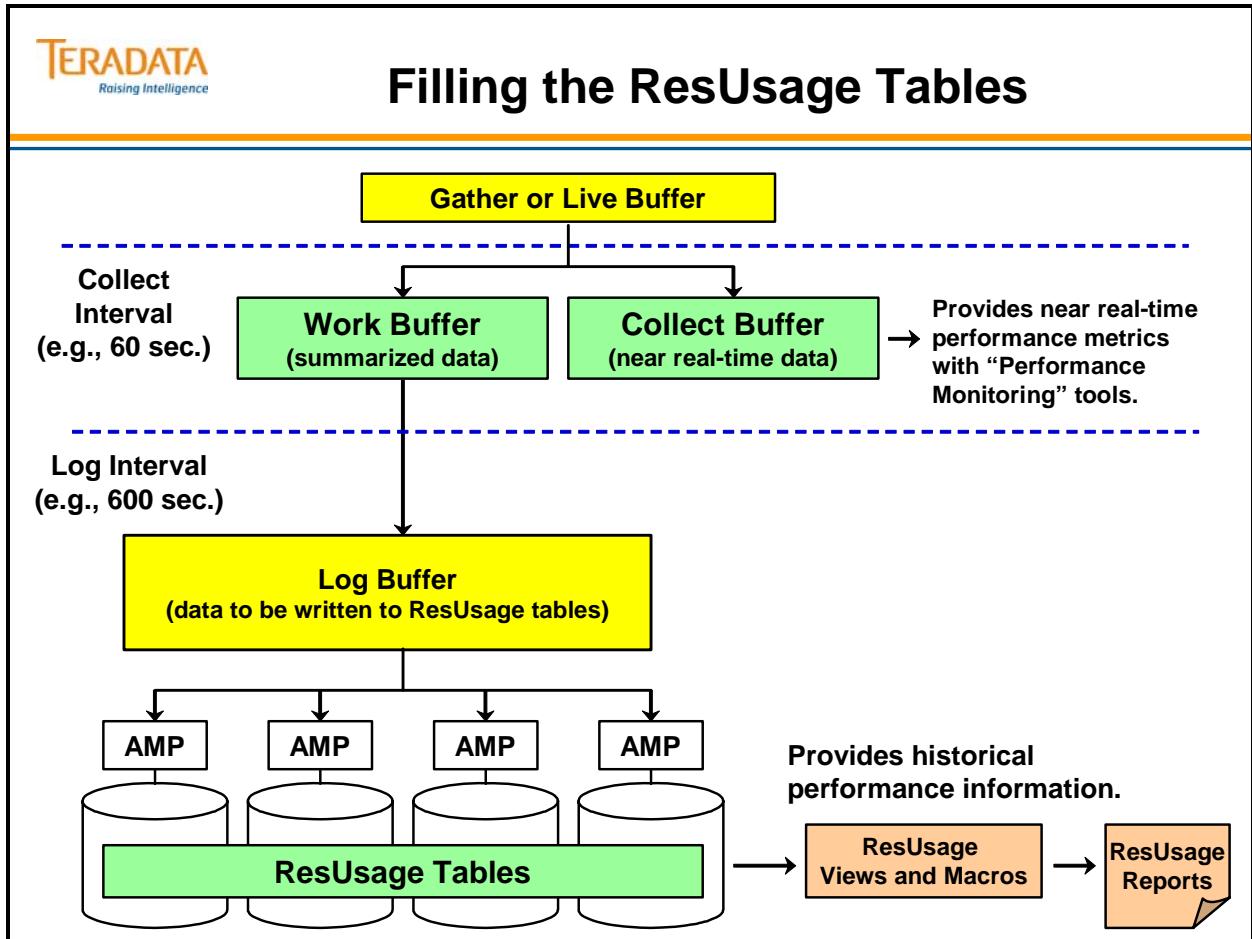
Data in ResUsage tables is after-the-fact data, because it has been collected and stored. With “Performance Monitor”, you view and capture data by looking directly into the collect buffer to see what is happening real-time on the system. This can be very useful for determining what is happening as a particular transaction or request is running.

Resource Usage Logging

When you initiate data logging, the system collects a variety of statistics for a period you specify. Teradata stores performance data in the DBC.ResUsage tables. Teradata uses nine tables to gather resource utilization data for a specified time period, and stores this information by node or vproc.

You can access the statistics stored in the DBC.ResUsage tables directly or use supplied views. In addition, you may create ResUsage reports using supplied macros that access the ResUsage views.

The system does not automatically collect resource utilization data. Consequently, you must activate resource collection and logging to gather performance data. Setting resource logging is described later in this module.



Specifying ResUsage Tables and Logging Rates

Various tools can be used to specify which of the ResUsage tables to collect and log information into. These tools include:

- xctl or ctl utility
- Teradata MultiTool Utility – CTL utility
- Supervisor commands (e.g., SET LOGTABLE ...) via DB Window

Note: Using the supervisor commands in the Database Window, you can enable tables and set collection and logging, but you cannot set Summary mode.

Specifying Tables using the xctl or ctl Utility

From a UNIX command line, enter the command: ***xctl -display hostname***. The command displays the xctl window.

To Set Node and Vproc Collect and Log Rates:

- From the View menu in the xctl window, click RSS Settings. The RSS Settings window opens.
- Enter the appropriate node and vproc rates.
- In the RSS Table Logging Enable section of the RSS Settings window, depress the button next to the name of each table you want enabled.
- In the RSS Summary Mode Enable section, depress the button next to the name of any enabled tables you want logged in summary mode.
 - When Summary Mode is active for table, one row is written to the database for each node in the system, summarizing all AMP data in each node, for each log interval.

Notes:

Rates can range from 0 – 3600 seconds (0 turns it off).

A log rate must be a multiple of the collect rate.

Specifying Tables using Supervisor Commands

You can also set resource logging from the Database Window Supervisor screen. The **set** and **get resource** commands may be used:

set logtable *tablename_or_ALL* ON/OFF

Example using MultiTool

The facing page illustrates selecting tables to collect and log via the MultiTool utility.



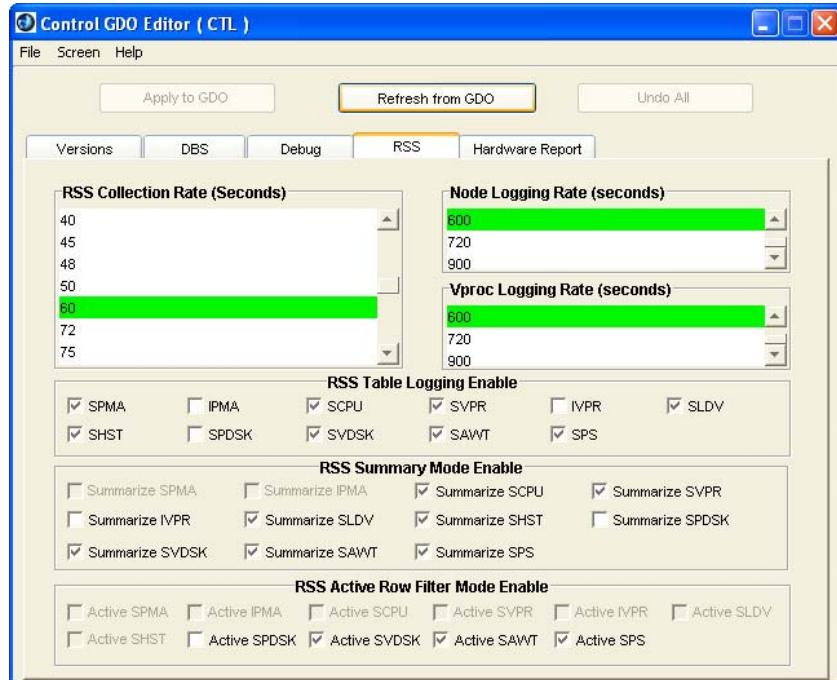
Specifying ResUsage Tables and Logging Rates

1st – Specify the collection and logging rates.

2nd – Specify which resource tables to collect and log.

Examples of tools that can be used to specify the tables to collect and log include:

- “control” utility – **ctl**
- Teradata MultiTool Utility – **CTL** utility
- Supervisor commands (e.g., **SET LOGTABLE ...**) via DB Window



Example of specifying tables to collect and log via MultiTool Version 12.0.

Resource Usage Tables

Teradata stores ResUsage data in a set of system tables. Each ResUsage macro derives its report from one or more of these tables. You must activate logging to produce a useful ResUsage report.

The Resource Sampling Subsystem (RSS) gathers ResUsage data through shared data collection buffers. The Collect buffer gathers entries according to the collection rate intervals. During the log rate interval, the entries are moved into the Log buffer. At the end of the log period, RSS will log the gathered data to the following ResUsage Tables and re-initialize the shared data collection buffers for the next log period.

All ResUsage table names begin with ResUsage and have the following extensions.

S: System or I: Internal (of interest mainly to Teradata development personnel)

pma:	node information
cpu:	cpu-specific information
vpr:	vproc information
ldv:	logical device information
hst:	Channel and LAN host information

The following tables are new starting with Teradata 12.0.

awt:	AMP Worker Task information
ps:	Priority Scheduler performance group information
pdsk:	AMP pdisk cylinder allocation, migration, and I/O statistics
vdsk:	AMP vdisk cylinder allocation, migration, and I/O statistics

Miscellaneous notes:

- All of the ResUsage tables are located in the DBC database.
- The DBC.ResUsageSobj table exists in V2R5, but is currently no used.

Setting Resource Logging from DBW

You can also set resource logging from the Database Window Supervisor screen. The **set** and **get resource** commands may be used:

```
set resource coll <vproc-collect-rate> vproc log <vproc log rate>
set resource coll <node-collect-rate> node log <node log rate>
get resource
```

Setting Resource Logging from the Control Utility – **ctl** or **xctl**

From a UNIX command line, enter the command: **xctl -display hostname**. The command displays the xctl window.



Resource Usage Tables

Node Data

ResUsageSpma	System-wide node information.	Generally enabled.
ResUsagelpma	System-wide internal node information.	Generally not needed.

CPU Data

ResUsageScpu	Information specific to the CPUs in a node.	Enable if Spma shows no obvious bottleneck.
---------------------	---	---

VProc Data

ResUsageSvpr	Data specific to each virtual processor	Generally enabled.
ResUsageVlpr	System-wide internal vproc information.	Generally not needed.

Host and LAN Data

ResUsageShst	Information specific to the host channels and LANs that communicate with TD.	Generally enabled.
---------------------	--	--------------------

Logical Device Data

ResUsageSldv	Logical Device – information specific to disk I/O.	Generally not needed.
---------------------	--	-----------------------

New with Teradata 12.0

ResUsageSawt	Collects and reports statistics about the AWTs.	
ResUsageSpdsk	Provides AMP-level Pdisk statistics.	Disable – currently not used.
ResUsageSvdsk	Provides AMP-level Vdisk statistics.	
ResUsageSps	Priority Scheduler Performance Group information.	

Resource Usage Views

Each row in the ResUsage tables represents activity during one logging period; the same is true of each row in the views. The difference between the tables and the views are the specific column values. ResUsage tables hold raw data. The views derive values from data in ResUsage tables.

ResGeneralInfoView

The ResGeneralInfoView provides an overview of system operation. Contains data from ResUsageSpma covering CPUs, disks, and BYNET information.

ResCPUUsageByAMPView

Contains data from ResUsageSvpr detailing the ways the CPUs are used by the AMPs.

ResCPUUsageByPEView

Contains data from ResUsageSvpr detailing the ways the CPUs are used by the PEs.

ResShstGroupView

The ResShstGroupView is based on the ResUsageShst table.

ResSldvGroupView

The ResSldvGroupView is based on the ResUsageSldv table.

ResSvdskGroupView (Teradata 12.0)

The ResSvdskGroupView is based on the ResUsageSvdsktable. This view includes resource usage detail on cylinder allocation, migration, and I/O statistics.



Resource Usage Views

ResUsage Tables

ResUsageSpma
ResUsageScpu
ResUsageSvpr
ResUsageShst
ResUsageSldv
ResUsageSawt*
ResUsageSps*
ResUsageSvdsk*

Calculations

(raw data)

The following types of data are derived from ResUsage tables:

- Resource utilization percentages
- Event counts per second
- Average event size

* Teradata 12.0

ResUsage Views

ResGeneralInfoView	View of general system information
ResCPUUsageByAMPView	View of CPU usage by AMP
ResCPUUsageByPEView	View of CPU usage by PE
ResShstGroupView	View of Host Channel and LAN activity
ResSldvGroupView	View of disk activity with Node Groups
ResVdskGroupView*	View AMP virtual disk activity

Resource Usage Macros

Resource usage macros produce reports from data collected in the resource usage tables. You can use the reports to analyze key operational statistics and evaluate the performance of your system. Like other macros, resource usage macros consist of one or more Teradata SQL statements stored in the Teradata Database and executed by a single EXECUTE statement.

In addition to the name of the macro, the EXECUTE statement for resource usage macros can include optional parameters to specify the following:

- A specific single node or a group of nodes
- Starting and ending dates and times
- Starting and ending nodes of a range of nodes

The macros are installed in the DBC database by DIP. You can run these macros after logging ResUsage data on a specific job or set of jobs.

There are different macros for one node, multiple nodes, a group of nodes, or all nodes. Examples of some of the macros are listed below.

Macro	Information Provided
ResCPUByAmp	CPU utilization of each AMP vproc.
ResCPUByPE	CPU utilization of each PE vproc.
ResCPUByNode	CPU utilization of the node.
ResHostByLink	Host traffic for each communication link.
ResLdvByNode	Logical device traffic channeled through the node by totaling its controller links into one summarized node output line.
ResMemMgmtByNode	Memory management activity information for the node.
ResNetByNode	Net traffic for the node.

Teradata features four ResNode macros that summarize resource usage.

Macro	Provides summary of ResUsage...
ResNode	Averaged across all nodes
ResOneNode	For a specific node
ResNodeByNode	Node by node
ResNodeByGroup	For a node grouping



Resource Usage Macros

The ResUsage facility provides macros to report information about Teradata.

One-node Macros	Multiple-node Macros	Group-node Macros	All-node Macros
ResCPUByAMPOneNode	ResCPUByAMP	ResAmpCpuByGroup	
ResCPUByPEOneNode	ResCPUByPE	ResPeCpuByGroup	
ResCPUOneNode	ResCPUByNode	ResCPUByGroup	
ResHostOneNode	ResHostByLink	ResHostByGroup	
ResLvdOneNode	ResLdvByNode	ResLdvByGroup	
ResMemMgmtOneNode	ResMemMgmtByNode	ResMemByGroup	
ResNetOneNode	ResNetByNode	ResNetByGroup	
ResOneNode	ResNodeByNode	ResNodeByGroup	ResNode
ResPdskOneNode	ResPdskByNode	ResPdskByGroup	
ResVdskOneNode	ResVdskByNode	ResVdskByGroup	
	ResPsByNode	ResPsByGroup	

Group-node macros are designed for co-existence systems.

* Teradata 12.0 macros

General Macro Syntax:

EXEC *Macroname* (*FromDate*, *ToDate*, *FromTime*, *ToTime* [, additional parameters depend on macro];

Example using ResNode macro:

EXEC DBC.ResNode (Date - 7, Date , ,);

This generates data from one week ago to today using the ResNode macro.

Example Output from DBC.ResNode Macro

The facing page contains an example of general ResUsage Summary information across all nodes.

There are 23 columns (including date and time) with the ResNode report. The first 5 columns (after the date and time) represent:

CPU Bsy%	Percent of time the CPUs are busy, based on average CPU usage per node
CPU Eff%	Parallel efficiency of node CPU usage. Parallel efficiency is the total percent of time nodes are busy. It is the average for all nodes of total busy divided by the total busy time of the busiest node.
WIO %	Percent of time the CPUs are idle and waiting for completion of an I/O operation.
Ldv IOs /Sec	Average number of logical device (disk) reads and writes per second per node.
Ldv Eff %	Parallel efficiency of the logical device (disk) I/Os. It is the average number of I/Os per node divided by the number of I/Os performed by the node with the most I/Os.

If a system's resources are (CPU and I/O) are heavily utilized, it may be necessary to add system resources. The DBC.ResNode macro can be used to provide before and after results.

The example on the facing page was captured for an NCR 5255 four-node system.



Example Output from DBC.ResNode

EXEC DBC.resnode ('2008-01-25', '2008-01-25', '08:00:00', '09:00:00');

Date	Time	CPU Bsy %	CPU Eff %	WIO %	Ldv IOs /Sec	Ldv Eff %	...
2008-01-25	08:00:00	98	100	0	1477	99	...
2008-01-25	08:10:00	99	100	0	1416	98	...
2008-01-25	08:20:00	100	100	0	1290	97	...
2008-01-25	08:30:00	100	100	0	1260	100	...
2008-01-25	08:40:00	95	99	1	1315	99	...
2008-01-25	08:50:00	97	100	0	1240	97	...

For the DBC.ResNode macro to display data, logging must be enabled on ResUsageSpma.

Notes about columns shown in this output:

- CPU Busy - % of time the CPUs are busy; based on average CPU usage per node.
- CPU Efficiency - parallel efficiency of node CPU usage; parallel efficiency is calculated by dividing average node utilization by maximum node utilization.
- WIO % - Percent of time the CPUs are idle and waiting for completion of an I/O operation.
- Ldv IOs /Sec - average number of logical device (disk) reads and writes per second per node.
- Ldv Eff % - parallel efficiency of the logical device (disk) I/Os.

PM/API and Performance Monitor

The PM/API (Performance Monitor/Application Programming Interface) facility is a real-time performance-monitoring tool that allows you to collect and return performance data on a Teradata Database with low overhead.

Teradata Performance Monitor (formerly PMON) is a Windows application (GUI) that uses the PM/API to provide real-time performance and session information.

How PM/API Collects Data

PM/API contains monitoring commands that you issue through a logon partition called MONITOR. MONITOR collects different types of performance data, including the current system configuration; resource usage and status of individual nodes or vprocs; and of individual sessions.

PM/API collects data in memory, not in a spool file on disk. As a result, PM/API routines (except the IDENTIFY command) cannot be blocked and consequently incur low overhead. PM/API stores node and vproc resource usage data and session-level usage data in separate collection areas. The data stored in memory is updated once during each sampling period. All users share the collected data.

The MONITOR partition collects and reports resource usage data differently from session-level usage data. To interpret the information that the MONITOR returns, you must understand the difference.

Collecting and Reporting Resource Usage Data

PM/API collects and reports node and vproc usage data for a single sample period. For example, a user sets the sampling period to 120 seconds. Then she issues the MONITOR RESOURCE request. The system collects node and/or vproc usage data during the next 120 seconds. If the user does not examine the data within the next 120 seconds, the data is lost when it is overwritten by data collected during the next 120-second interval.

Collecting and Reporting Session-level Usage Data

PM/API cumulatively collects session-level usage data, such as counts and “time used.” Other data, such as locking information and “AMP State,” is not gathered cumulatively. The sampling period limits how frequently the cumulative data is updated.

Example

A user sets the sampling period to 300 seconds and issues the MONITOR SESSION request. The system collects new information every 300 seconds, and adds the information to the existing total in a cumulative fashion. Session-level data includes data for the beginning 300 seconds as well as for any subsequent intervals.



PM/API and Performance Monitor

The PM/API (Performance Monitor/Application Programming Interface) ...

- is part of Teradata software and has low overhead.
- provides real-time monitoring capability and session information.
- provides the following data.
 - **Processor Data**
 - Collects/reports node/vproc usage for single period.
 - New period overwrites data from previous period.
 - Collection is not cumulative.
 - **Session-level Data**
 - Collects/reports session-level data cumulatively.
 - New sampling period increases collected data.
- Accessed via customized application or Teradata Performance Monitor application

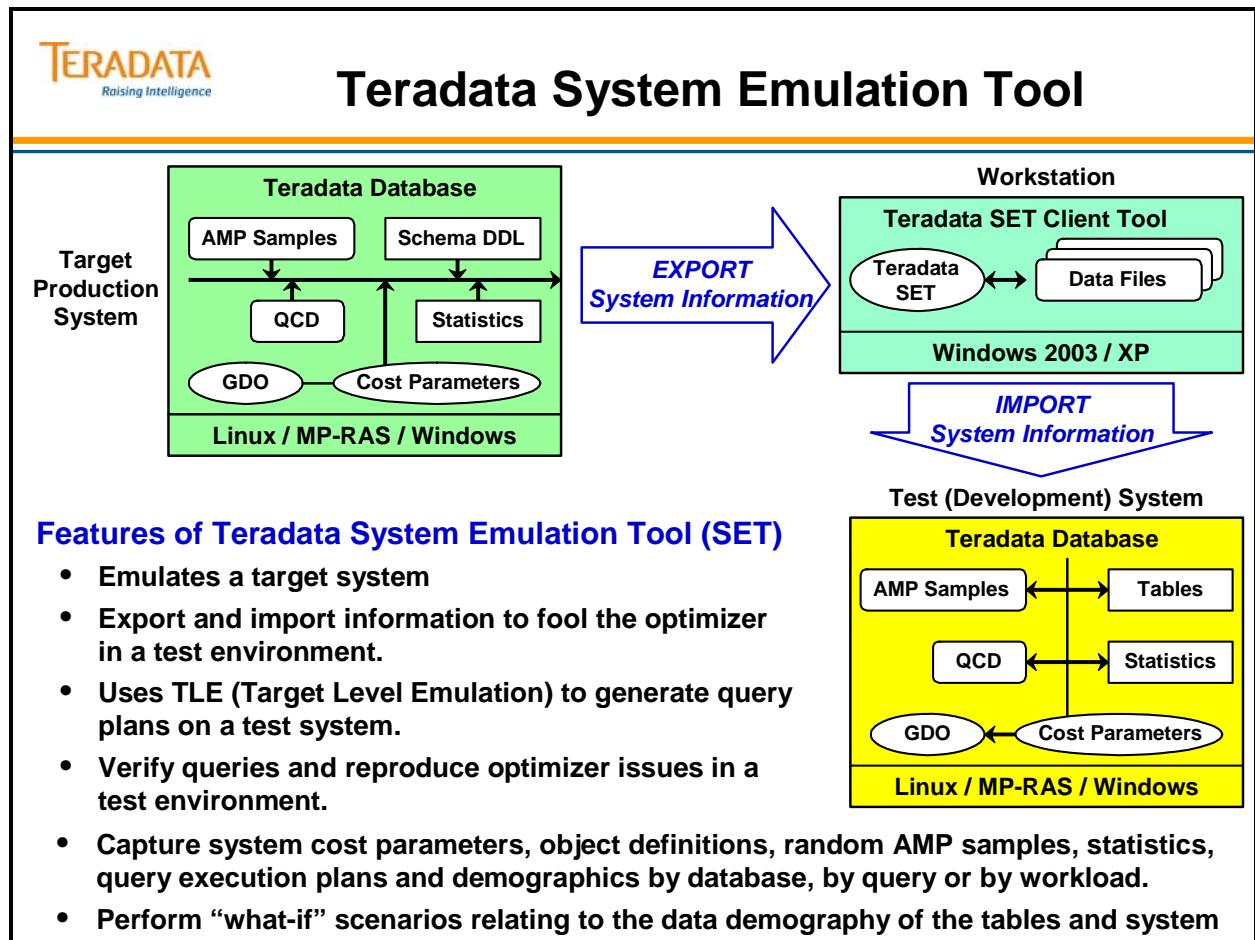
Teradata Performance Monitor

- Windows GUI application that utilizes the PM/API and can be used to ...
 - provide real-time performance monitoring.
 - show how efficiently the Teradata RDBMS is using its resources.
 - identify problem sessions and users.
 - abort sessions and users having a negative impact on system performance.

Teradata System Emulation Tool (Teradata SET)

The Teradata System Emulation Tool simplifies the task of emulating a target system by providing the ability to export and import all of the information necessary to fake out the optimizer in a test environment. This information can be used along with Teradata's Target Level Emulation feature to generate query plans on the test system as if they were run on the target system. This feature is useful for verifying queries and reproducing optimizer related issues in a test environment.

Teradata SET allows the user to capture system cost parameters, object definitions, random AMP samples, statistics, query execution plans and demographics by database, by query or by workload. This tool does not export user data. Upon import the user can customize or edit object definitions, random AMP samples, statistics and cost parameters. The Customize feature allows the user to perform “what-if” scenarios relating to the data demography of the tables and system performance parameters. Teradata SET also has an option to log SQL statements. The user can view the log directly from the Teradata SET window to troubleshoot any failures that occur during export or import operations.



Performance Monitoring Summary

The facing page summarizes some important concepts regarding this module.



Performance Monitoring Summary

- Resource usage ([ResUsage](#)) data and reports can help you to improve system performance and management.
- Various tools exist to specify the tables to collect and log as well as setting the collection and log rates.
- You can access the ResUsage statistics stored in the ResUsage tables directly or use supplied views and macros.
- The [PM/API facility](#) is an application programming interface that can be used to provides a real-time monitoring capability and session information.
- [Teradata Performance Monitor](#) is a Windows GUI application that utilizes the PM/API

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. What are two methods of setting collection and logging intervals?

2. Match the following tools to its description.

- | | |
|--|---------------------------------------|
| <input type="checkbox"/> 1. ResUsage tables | A. Set collection and log rates |
| <input type="checkbox"/> 2. Teradata Performance Monitor | B. Emulates a target system |
| <input type="checkbox"/> 3. Teradata SET | C. Provides session level information |
| <input type="checkbox"/> 4. ctl | D. Holds historical resource data |

Teradata Training

Notes

Module 53



Utilities Overview and Teradata DB Window

After completing this module, you will be able to:

- Describe the three ways to initiate AMP-based utilities.
- List two techniques to access the Supervisor function of the Console task.
- Identify the purpose of various DBS Control utility parameters.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Locations from which Utilities can be Executed	53-4
Host-based Utilities	53-4
AMP-based Utilities.....	53-4
Example of Utilities available through HUTCNS:	53-4
Examples of Utilities available through the Teradata DB Window:.....	53-4
SMP and Database Window Utilities.....	53-6
Teradata Console Task.....	53-8
cnstool and cnsterm commands	53-8
Starting cnstool	53-8
Starting cnsterm	53-8
Teradata Database Window	53-10
Database Window Icons.....	53-10
DBW Supervisor Window	53-12
Sub-windows.....	53-12
Teradata MultiTool (Windows and Linux).....	53-14
PDE State Arrow.....	53-14
DBS State Arrow	53-14
Applications (Tools) that can be started via MultiTool	53-14
DB Window via MultiTool	53-16
Accessing Console Utilities via the DB Window	53-18
DBS Control Utility	53-20
DBS Control Record – General Fields (V2R5.1)	53-22
DBS Control Record – General Fields (TD 12.0).....	53-24
DBS Control Record – File System Fields	53-26
DBS Control Record – Performance Fields	53-28
DBS Control Record – Checksum Fields.....	53-30
Modifying DBS Control Parameters.....	53-32
Example: Set the Century Break value	53-32
Summary	53-34
Review Questions	53-36

Locations from which Utilities can be Executed

The Teradata software includes two different types of utilities: host-based utilities and AMP-based utilities.

Host-based Utilities

Host-based utilities are programs/applications that you install on a host system. The term “host” may refer to a channel-attached host or a network-attached host. Host-based utilities run under the host operating system.

The Archive and Recovery Utility (ARC) and DUMP Unload/Load Utility (DUL, DULtape) are discussed later in this course.

AMP-based Utilities

You initialize AMP-based utilities using the Teradata Database Window. A console interface called Host Utility Console (HUTCNS) is a host-based utility that runs on a channel-attached mainframe and provides access to a number of AMP-based utilities.

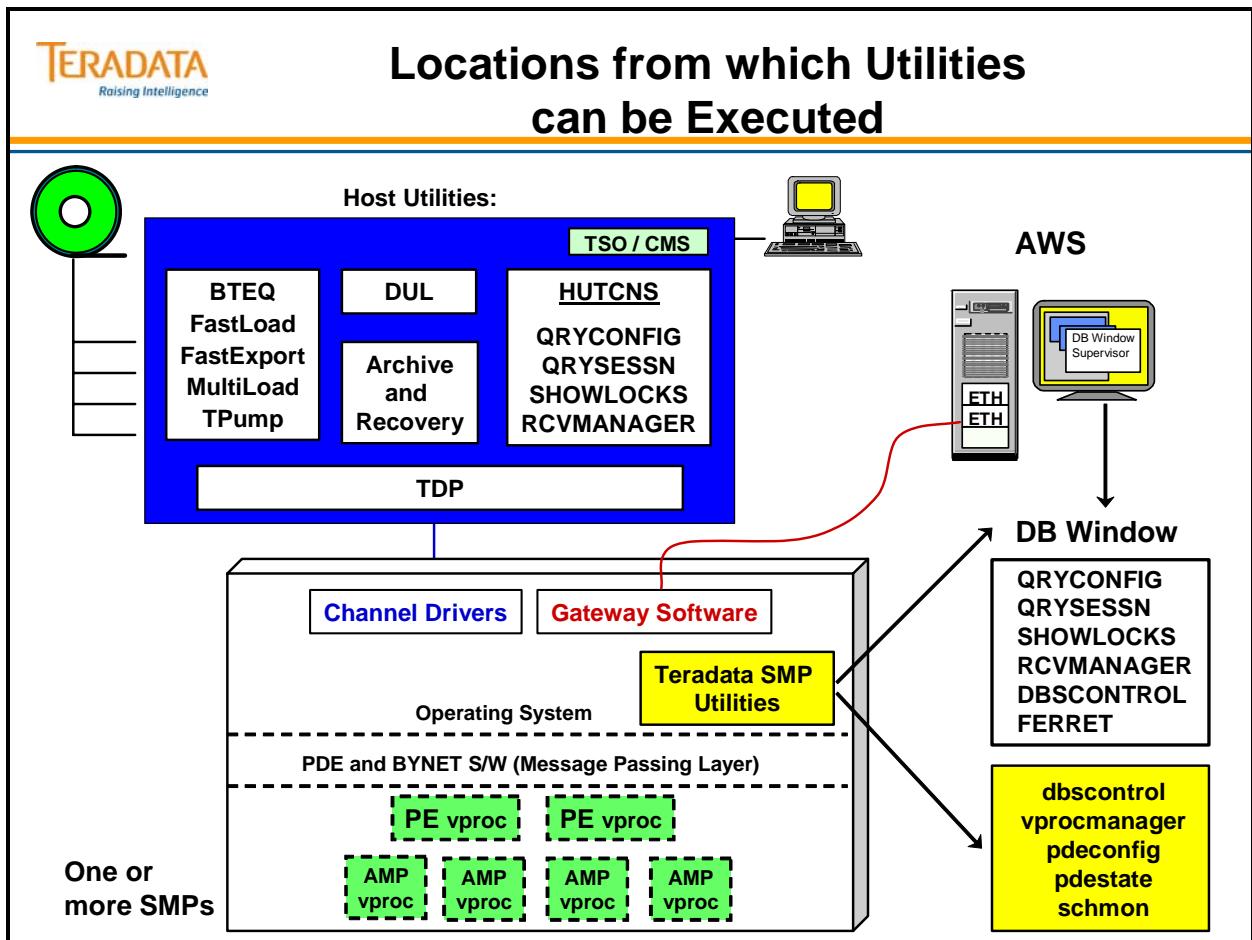
The diagram on the facing page shows AMP-based utilities available through HUTCNS on a channel-attached mainframe and those available through the database window.

Example of Utilities available through HUTCNS:

Session Status	(QRYSESSN)	enter SES
Configuration Display	(QRYCONFIG)	enter CON
Locks Display	(SHOWLOCKS)	enter LOC
Recovery Manager	(RCVMANAGER)	enter RCV

Examples of Utilities available through the Teradata DB Window:

- | | |
|---------------------------|---|
| Ferret → Showspace | Displays, by AMP vproc or by Vdisk, the number of physical cylinders in use for permanent table storage and spool, as well as the average percent cylinder utilization. |
| Ferret → Packdisk | Packs system and user data stored on permanent data cylinders to a specified density. You can select one or more vprocs to pack. |
| Abort Host | Aborts all outstanding transactions running on behalf of a host that has crashed. |



SMP and Database Window Utilities

The facing page lists various Teradata control and support utilities and from where they are executed.

These functions can be secured through the SMP /**ntos/cnsperms** and /**ntos/rhosts** files.
This /**ntos/cnsperms** file contains 4 fields separated by colons. The format is:

A:B:C:D where

A = unix_userid_name@hostname

B = ALL or a list of allowed supervisor commands

C = Utilities the user is allowed to start

D = Utilities the user is NOT allowed to start

List of commands or utilities is space separated.

Field D is ignored if field C is present.

This file is maintained by DB Window Supervisor commands.

- GET PERMISSIONS
- GRANT
- REVOKE

This file is automatically copied to all SMP nodes by PDE software. Only execute these commands when all SMP nodes are available so that all copies are updated.

Note that the HELP and GET commands require no permissions (these are always accepted).

SMP utilities are typically found in the following UNIX Directories:

- /usr/bin (ex., bteq, fastload, ...)
- /usr/ntos/bin (ex., cnsterm, tpareset, ...)
- /tpasw/bin (ex., dbscontrol, ...)
- /sbin (ex., schmon, ...)



SMP and Database Window Utilities

SMP – UNIX, Windows, or Linux

Command Line (Examples)

tpareset
pdestate
dbctrl
vprocmanager
cnsterm (UNIX only)
cnstool
schmon
dip

X Windows

xdbw
xctl
xgtwglobal
xperfstate

Miscellaneous Utilities

sysadm (UNIX)
Teradata MultiTool (Windows)
Teradata Administrator
BTEQ
SQL Assistant
Load utilities (e.g., FastLoad)
Teradata Manager
Remote Console
PSA

Windows

dbw
ctl
gtwglobal

Supervisor

ABORT SESSION
CNSGET
CNSSET [options]
DISABLE LOGONS
ENABLE LOGONS
GET CONFIG
GET LOGTABLE
GET PERMISSIONS
GET RESOURCE
GET SSO
GET TIME
GET VERSION
GRANT
HELP
LOG
QUERY STATE
RESTART TPA
REVOKE
SET LOGTABLE
SET RESOURCE
SET SESSION COLLECTION
START **
STOP

Teradata DB Window

** START Application

ABORTHOST
CHECKTABLE
CONFIG
DBSCONTROL
DIP
FERRET
FILER
LOKDISP
QRYCONFIG
QRYSESSN
REBUILD
RECONFIG
RCVMANAGER
SHOWLOCKS
UPDATESPACE
VPROCMANAGER

FERRET Utilities

DEFragment
PACKDISK
SHOWBLOCKS
SHOWSPACE
SCANDISK
SCOPE
**UPDATE DATA INTEGRITY
(V2R5.1 Function)**

Teradata Console Task

The Teradata Console (CNS) task is responsible for managing the Teradata DB Window.

There are 3 ways in which the CNS task can be invoked.

- Xdbw (Teradata Database window)
- /usr/ntos/bin/cnsterm (command-line interface)
- /usr/ntos/bin/cnstool (command-line interface)

cnstool and cnsterm commands

cnstool and **cnsterm** are command-line interfaces to the Teradata DB Console functions. **cnstool** is available with both UNIX and Windows 2000 systems. **cnsterm** is only available with UNIX MP-RAS systems.

Starting cnstool

After executing **cnstool**, commands directed to the Supervisor or any application area have to be preceded with the appropriate number. Window numbers 1 through 4 are the console utility windows, 5 is the Database I/O window, and 6 is the Supervisor Window.

Only the root user is allowed to use this command: # **cnstool**

For example, to enter a Supervisor command such as get version: **6:get version**

To start a utility such as qrysessn, enter the following:

6: start qryconfig (assume qryconfig is started in area 1)
1:offline; (offline is a qryconfig option)

To exit from cnstool, enter either **Del** or **Control C**.

Starting cnsterm

When you start cnsterm, the only command line option is the window number.

Only the root user is allowed to use this command: # **cnsterm n**

where **n** is the window number.

For example, to display the supervisor: # **cnsterm 6**

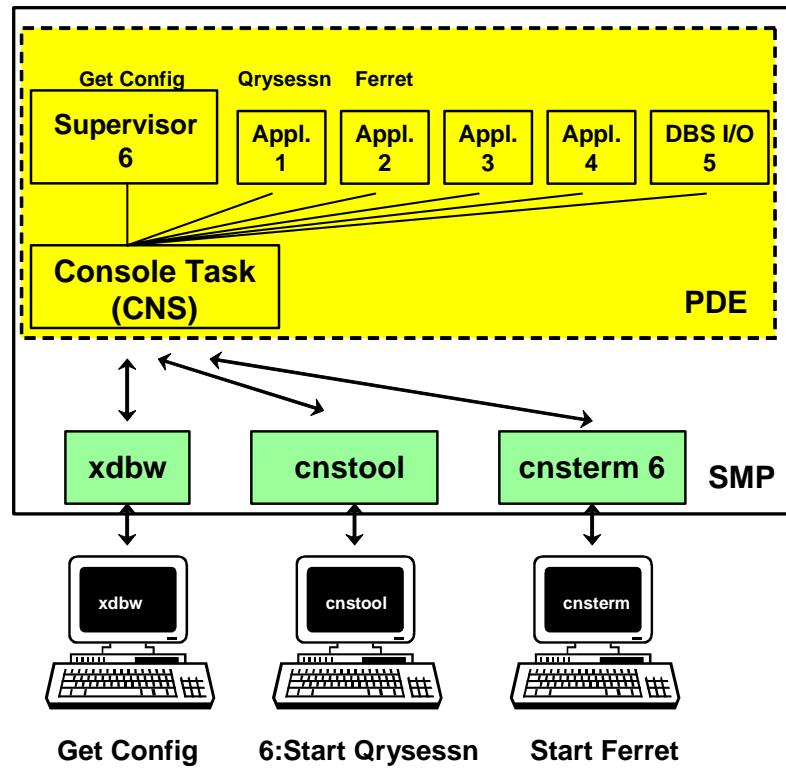
To exit from any screen, press the keyboard's "break" or interrupt key. (You can check your interrupt key setting with the UNIX command, stty.) The interrupt key is often set to **Del** or **Control C**



Teradata Console Task

The Supervisor is accessed via ...

- 1 - xdbw (Database Window)
- 2 - cnstool (command line)
- 3 - cnsterm (command line - UNIX MP-RAS only)



Teradata Database Window

The Database Window (DBW) is the console software for the Teradata Database. The DBW software provides a dimension of flexibility to a database administrator since you can start the console from virtually any workstation.

Database Window Icons

The DBW contains an icon labeled “Supvr” that opens the Supervisor window. You can start Teradata AMP-based utilities from the Supervisor window.

Once a Teradata utility is running, the DBW displays a new application icon. This icon opens a window where the Teradata utility is running. You can have a number of application icons present in the DBW, with each representing a different Teradata utility. You may move back and forth from one utility to another by returning to the DBW.

You can have multiple instances of the DBW window running at the same time. While you can have up to nine DBW windows open, you probably should not have more than 7 open. Two windows should be reserved for remote support, if necessary.

There are four application utility partitions available with the DBW window. You can run up to four utilities at one time, as well as any commands you execute from the Supervisor window.

To start DBW in UNIX, execute the following command from the UNIX command line:

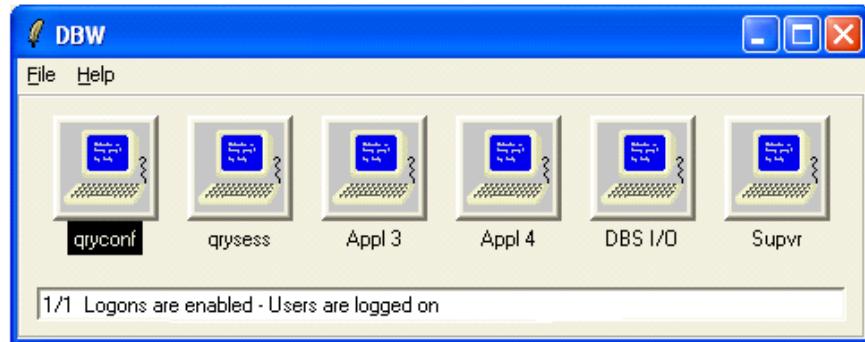
/usr/ntos/bin/xdbw -display hostname:0 &

You must enter the command from the PDN node, or you must specify the PDN node using the machine option with the command as shown:

i -machine I7544 -display hostname:0 &



Teradata Database Window



Database Window (DBW)

- The Database Window is the console software for Teradata Database Version 2 implementations.

Database Window Icons

- The Database Window contains a “Supvr” icon that opens the Supervisor window.
- Once a Teradata utility is running, the DBW displays a new icon which opens a window for the Teradata utility.

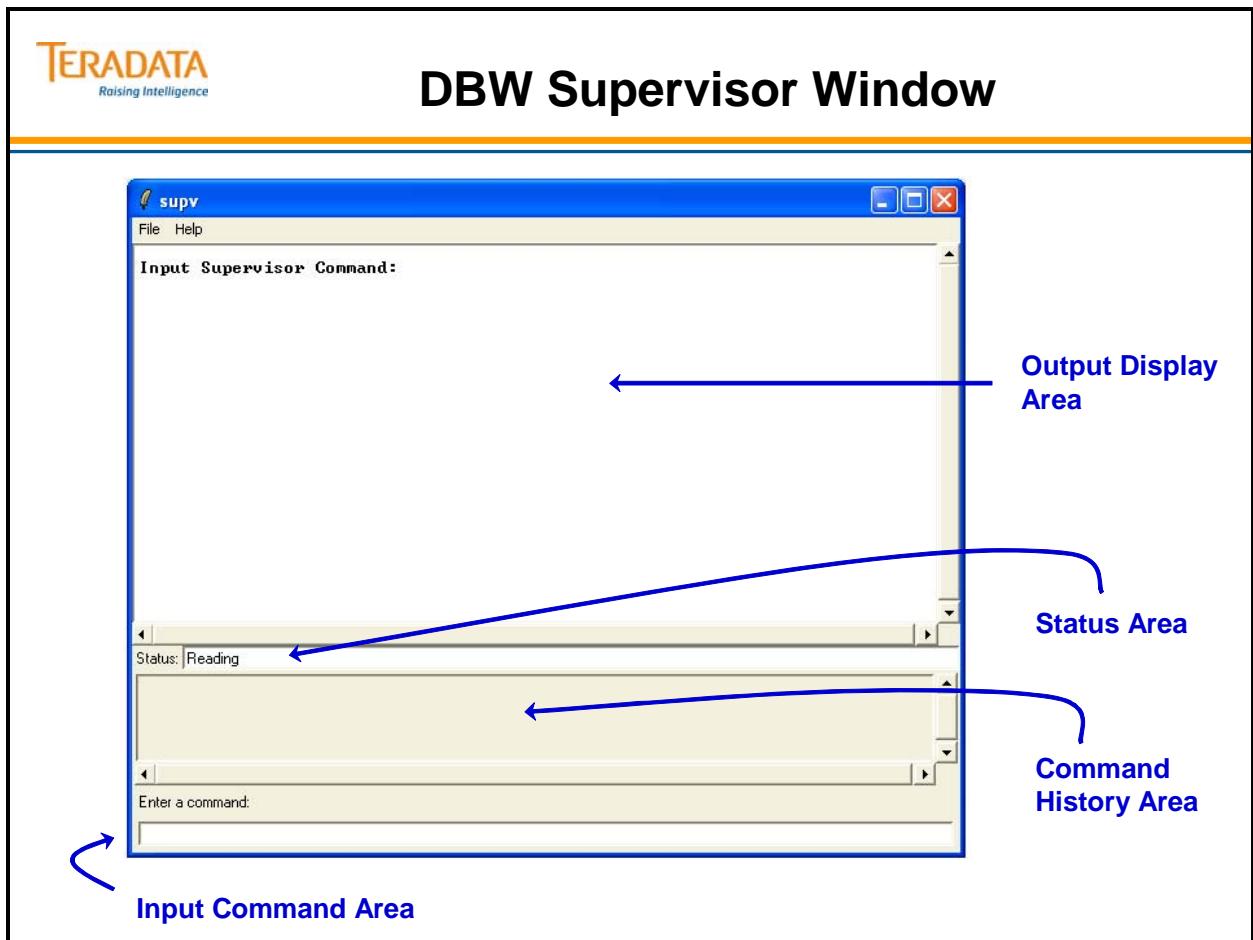
DBW Supervisor Window

You must start the DBW before you can start the Supervisor program. To open the Supervisor window, click the “Supvr” icon in the DBW.

Sub-windows

The Supervisor window contains the following four sub-windows:

Output	Displays the results of user commands. It displays <i>Input Supervisor Command</i> when the Supervisor window first opens. Use the scroll bars to review results from previous commands not currently visible in the Output sub-window.
Status	Displays the current status message. The word <i>Status:</i> appears to the left of the sub-window and is used by CNS to indicate the state of the application running in this window. Current states include: Blank, Running: and Reading:
Command History	Displays a list of commands you previously entered. Use the scroll bars to review commands previously entered that are not currently visible in the Command History sub-window.
Input	Area where you type commands. The phrase <i>Enter a command:</i> appears just above this sub-window.



Teradata MultiTool (Windows and Linux)

Teradata MultiTool provides a Graphical User Interface (GUI) on Windows and Linux that provides Teradata administrators and support personnel with a Windows interface to command-line-based Teradata and PDE tasks. The interface is additional to the existing user interfaces.

Teradata MultiTool is a Graphical User Interface (GUI) that you can use to start specific utilities. You can also start many utilities from the Supervisor Window within Teradata MultiTool.

To start Teradata MultiTool, do the following:

Start → Programs → Teradata RDBMS → Teradata MultiTool

PDE State Arrow

State information is received when you execute a pdestate -a command. In general, the following applies:

- An upward-pointing green arrow indicates a component is UP.
- A downward-pointing red arrow indicates a component is DOWN.
- An animated green or red arrow indicates that the PDE is in transition.

DBS State Arrow

In the DBS area, the following applies:

- An upward-pointing green arrow indicates a component is UP.
- A downward-pointing red arrow indicates a component is DOWN.
- An animated green or red arrow indicates that the Teradata RDBMS is in transition.

In addition, an upward-pointing arrow indicates the following:

- Green indicates that logons are enabled.
- Red indicates that logons are disabled.
- Three faces indicate that users are logged on. If no users are logged on, the faces are not present, and the RDBMS is quiescent.

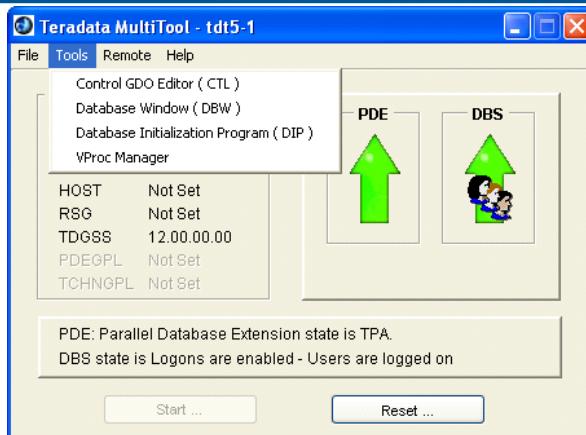
Applications (Tools) that can be started via MultiTool

- Control GDO Editor (CTL) display and modify the fields of the PDE GDO (Globally Distributed Object).
- Database Window (DBW) - activate the Teradata DB Window
- Database Initialization Program (DIP) executes one or more of the standard Database Initialization Program (SQL) scripts packaged with Teradata.
- Vproc Manager – numerous support functions such as change vproc states, initialize and boot a specific vproc, etc.



Teradata MultiTool (Windows and Linux)

Example of Main Window of MultiTool



The Teradata MultiTool utility (Windows and Linux) can be used to ...

- Start and reset Teradata
- Check status and versions of Teradata software
- Initiate various support applications
 - Control GDO Editor (CTL) - same function as xctl
 - Database Window (DBW) - activate the Teradata DB Window
 - Database Initialization Program (DIP) scripts
 - Vproc Manager - change state of vprocs, initialize a Vdisk, restart Teradata
- Connect to a remote Teradata node

DB Window via MultiTool

The DBW may also be started via the MultiTool program.

In the Teradata MultiTool main window, select Tools -> Database Window (DBW) and the DBW should appear.

You can create files that log all output that appears in the Supervisor window or any of the application windows. These logs might be useful when you want to review or print information.

When you log all windows, standard log files are opened. If a log file already exists, the system overwrites the old log with the new log.

To start a log, Select the “File” menu and then select “Enable Logging”. The Select Logging File dialog appears and complete the dialog box.

The following list identifies the default file names, which are located in the *drive*:\\Program Files\\NCR\\LPDE directory, where *drive* is the drive where you installed the default files.

- Supervisor – SupvLog
- Application window 1 – App1Log
- Application window 2 – App2Log
- Application window 3 – App3Log
- Application window 4 – App4Log

TERADATA
Raising Intelligence

DB Window via MultiTool

This example shows the Teradata DB Window supervisor function started via MultiTool.

The various functions appear as tabs within the window.

This example shows execution of a simple command (get config) and starting a utility.

Database Window (DBW)

File Help

Supervisor Application 1 Application 2 Application 3 Application 4 DBS IO

ID	State	Number CPUs (MB)	CHANs LANs AMPs	Node Name
1-048	ONLINE	0 2 4009	0 1 7	byn001-4
1-05	ONLINE	0 2 4009	0 1 7	byn001-5

0 PDE Distribution Node
PDE State: TPA
Input Supervisor Command:
start dbscontrol
Started 'dbscontrol' in window 1
at Mon Feb 4 13:17:38 2008
Input Supervisor Command:

Status: Reading

get config
start dbscontrol

Command: start dbscontrol

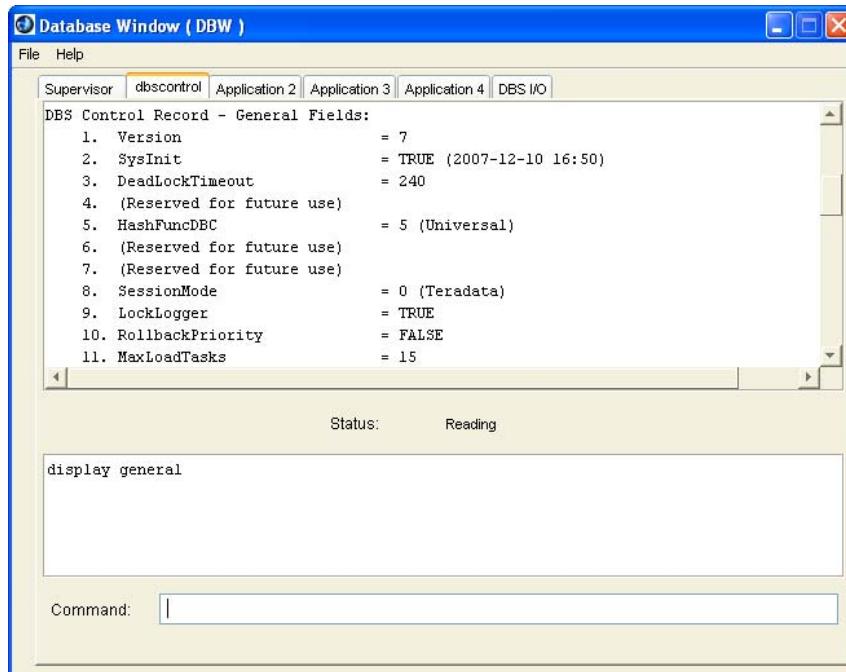
From Supervisor, enter: START DBSCONTROL

Accessing Console Utilities via the DB Window

The facing page shows an example starting the DBS Control utility from the supervisor and the initial screen of this utility.



Accessing Console Utilities via the DB Window



DBSCONTROL utility is started in a separate tab.

DBS Control Utility

The DBS Control utility is used to view/modify the DBS Control record fields which:

- Establish system values
- Tune performance
- Debug/diagnose problems

DBSControl Record fields are logically grouped based on how the Teradata Database uses them. The physical position of the field in the record is not significant. The group names are defined as follows:

This group...	Contains fields used...
General	For various purposes by the Teradata Database
File System	By Teradata Database V2 File System
Performance	For tuning performance
CheckSum	Set of parameters for CHECKSUM feature

There are three ways to access the DBSControl utility:

1. DB Window – **START DBSCONTROL**
2. Command line – **/tpasw/bin/dbscontrol**
3. Teradata Manager – Remote Database Console

Because modifying the DBSControl Record fields may have system wide ramifications, only trained personnel should use it.

For more information about DBSControl and all fields displayed by the utility, see the *Teradata RDBMS User Utilities*



DBS Control Utility

The DBS Control utility is used to view/modify the DBS Control Record fields which:

- Establish system values
- Tune performance
- Debug/diagnose problems

There are multiple ways to access the DBS Control utility.

1. DB Window (Supervisor) – START DBSCONTROL
2. Teradata Manager – Remote Database Console
3. Command line – /tpasw/bin/dbscontrol

The DBS Control Record parameters are divided into four categories:

- General
- File System
- Performance
- Checksum (new with V2R5.1)

DISPLAY or HELP commands

- DISPLAY GENERAL | FILESYS | PERFORMANCE | CHECKSUM
- HELP GENERAL | FILESYS | PERFORMANCE | CHECKSUM

DBS Control Record – General Fields (V2R5.1)

This page only lists the first 27 general parameters.

Field	Purpose
1. Version	Indicates the DBS Control Record version number set by SysInit. Cannot be changed with DBSControl.
2. SysInit	Indicates whether or not the last SysInit was successful – cannot be changed with DBSControl. - set by SysInit.
3. DeadLockTimeout	Used for deadlock time-out detection cycles – time is in seconds.
4. Reserved	Reserved for future use.
5. HashFuncDBC	Defines the DBS hashing function. Cannot be changed with DBSControl - set by SysInit.
6. Reserved	Reserved for future use.
7. Reserved	Reserved for future use.
8. SessionMode	This field defines the system default transaction mode, case sensitivity, and character truncation rule for a session. The setting is either 0 (Teradata) or 1 (ANSI).
9. LockLogger	Enables logging of lock delays due to database locks.
10. Rollback Priority	Defines the priority for rollback operations – RUSH or user priority. If RUSH is wanted, set to TRUE.
11. MaxLoadTasks	Controls the combined number of FastLoad, MultiLoad, and FastExport tasks allowed (max is 15).
12. RollForwardLock	Defines the system default for the RollForward Using Row Hash Locks option. This allows the DBA to specify that row hash locks should be used to lock the target table rows during a RollForward. To enable this feature set the field to TRUE. To disable the feature set the field to FALSE.
13. MaxDecimal	Defines maximum number of decimal digits used in expression typing (valid values are 15 and 18).
14. CenturyBreak	This field defines which Teradata dates are specific to the 21st Century. For example, C = 25, strings inserted such as 00/01/01 and 24/01/01 will assume the years 2000 and 2024 respectively. A string inserted as 25/01/01 will assume the year 1925. Valid values range from 0 to 100.
15. DateForm	Defines the standard date format – IntegerDate is 0, ANSIDate is 1.
16. System TimeZone Hour	This field defines the System TimeZone Hour offset from UTC. Permitted range: -12 to 13.
17. System TimeZone Minute	This field defines the System Time Zone Minute offset from UTC. Permitted range: -59 to 59.
18. RollbackRSTransaction	Used when a subscriber replicated transaction and a user transaction are involved in a deadlock. TRUE means rollback the subscriber replicated transaction. FALSE means rollback the user transaction.
19. RSDeadLockInterval	Deadlock checking between a subscriber-replicated transaction and a user transaction.
20. RoundHalfwayMagUp	This field indicates how rounding should be performed when computing values of DECIMAL type.
21. DefaultDateFormat	Default date format for the system. If a date format is defined then the format is used as the default for the system. Date formats are defined between enclosing single quotes ‘’. The following is an example, m g 21 = ‘yyyy/mm/dd’.
22. Target Level Emulation	Used when a user wants to emulate a Target Level Machine. TRUE means the user wants to run Target Level Emulation. FALSE means the user does NOT. The default is FALSE
23. Export Width Table ID	For Expected Defaults, enter 0. For Compatibility Defaults, enter 1. For Maximum Defaults, enter 2.
24. EnableStepText	When TRUE, dispatcher step text will include names and costs. When FALSE no name and cost information will be available. The default value is FALSE.
25. EnableDBQM	When TRUE, validation of all SQL through the DBQM rule will be enforced. When FALSE no validation through DBQM will be done. This toggle has a dependency on the EnableStepText toggle (#24). The default value is FALSE.
26. External Authentication	26. - This field indicates whether external authentication is enabled. The valid values are 0, 1 and 2. 0: Accept both external authentication and traditional logons. Default value is 0. 1: Reject external authentication and accept traditional logons. 2: Accept external authentication and reject traditional logons.
27. IdColBatchSize	Indicates the size of the pool of numbers to be reserved for generating numbers for a batch of rows to be bulk-inserted into a table with an identity column. The valid range of value is 1 .. 1000000. The default value is 100000.



DBS Control Record – General Fields (V2R5.1)

General Fields: (these are the defaults)

1. Version	= 5	21. (Reserved for future use)	
2. SysInit	= TRUE	22. Target Level Emulation	= FALSE
3. DeadLockTimeOut	= 240 (seconds)	23. Export Width Table ID	= 0 (Expected Defaults)
4. (Reserved for future use)		24. (Reserved for future use)	
5. HashFuncDBC	= 5 (Universal)	25. (Reserved for future use)	
6. (Reserved for future use)		26. Single Sign On	= 0 (On)
7. (Reserved for future use)		27. IdCol Batch Size	= 100000
8. SessionMode	= 0 (Teradata)	28. LockLogger Delay Filter*	= FALSE
9. LockLogger	= FALSE	29. LockLogger Delay Filter Time*	= 0
10. RollbackPriority	= FALSE	30. ObjectUseCountCollectRate*	= 0 minutes (Disabled)
11. MaxLoadTasks	= 5		
12. RollForwardLock	= FALSE		
13. MaxDecimal	= 15		
14. CenturyBreak	= 0		
15. DateForm	= 0 (IntegerDate)		
16. System TimeZone Hour	= 0		
17. System TimeZone Minute	= 0		
18. RollbackRSTransaction	= FALSE		
19. RSDeadLockInterval	= 0 (240)		
20. RoundHalfwayMagUp	= FALSE		

* New or changed with V2R5.1.

DBS Control Record – General Fields (TD 12.0)

This page only lists the additional general parameters.

Field	Purpose
28. LockLogger Delay Filter	Indicate whether locking logger delay filter is ON. If it is ON, the delay filter time in field 29 will take effect. The default is OFF.
29. LockLogger Delay Filter Time	Indicates the delay filter time used by locking logger. Only blocked lock request with delay time greater than this value will be logger. The valid range is 0 .. 1000000 seconds, default is 0 second.
30. Object Use Count Collect Rate	Determines the default amount of time which the Data Dictionary columns of database object AccessCount and LastAccess Time Stamp are reset automatically. The default is 0, which disables the database object use count feature. To enable, the recommended value is 10 minutes.
31. LockLogSegmentSize	Indicates the size of the locking logger segment used by the lock manager in kilo bytes. The minimum value is 64 KB and maximum value is 1024 KB.
32. System Default Cost Profile Id.	This value is used to select the cost profile from DBC.CostProfiles that will be used to create cost prediction method parameter values. The profile with DBC.CostProfiles.ProfileId == CostProfileId will be chosen. If CostProfileId doesn't match any ProfileId value, then the system default profile (DBC.CostProfiles.ProfileId == 0) is used. If the system default profile cannot be read then the bootstrap profile is used.
33. DBQLFlushRate	Determines the frequency for writing DBQL cache entries to DBQL dictionary tables. Default is 600 seconds (10 minutes). Valid range is 1 to 3600 seconds. The recommended value is 600 seconds or more.
34. Memory Limit Per Transaction	Specifies the maximum amount (number of pages) of in-memory temporary storage that can be used by the RSG to store the records for one transaction. If the transaction exhausts this amount, then it is moved to a disk file (specified by general field number 37). Default - 2 pages; minimum - 0 pages and maximum value is 127 pages.
35. Client Reset Timeout	Specifies how long the RSG should wait for an intermediary to reconnect after a communication failure, an intermediary reset, or a server reset before taking the needed actions. Default - 300 seconds; minimum - 0 sec and maximum value is 65535 sec.
36. Temporary Storage Page Size	Specifies the size (in KB) of a memory page that is used to store data rows of a replicated transaction. Default - 4 KB; minimum - 1 KB and maximum is 1024 KB.
37. Spill File Path	Specifies a directory that will be used by the RSG for spill files. The default path is C:\Program Files\NCR\Tdat\tdconfig\tdrsg
38. MDS Is Enabled	Assumed that the MDS packages are installed, the MDS will be running if this flag is TRUE. The default is FALSE.
39. Checktable table lock retry limit	(Default = 0 is retry forever). When a table is locked, checktable will retry until it gets the table. If the retry limit is set, checktable will retry within the specified limit. Retry limit range = 0 to 32767 minutes
40. EnableCostProfileTLE	This boolean determines whether new Optimizer Cost Profile System (OCES) diagnostics are enabled in combination with Target Level Emulation (TLE). This is initialized to FALSE. Attempts to use combined OCES and TLE diagnostic syntax will get a syntax error. When set to TRUE, combined OCES/TLE diagnostic syntax is enabled. When enabled all TLE diagnostics are also processed by the new OCES logic. If EnableSetCostProfile == 0 or == 20, then this field will be set FALSE.
41. EnableSetCostProfile	This controls usage of DIAGNOSTIC SET PROFILE ... statements to dynamically change cost profiles used for query planning. Meaningful values are 0,1,2,20,21, and 22.
42. UseVirtualSysDefault	If this == 0, then the system default cost profile is always SysDefault. If this > 0, then the system default cost profile is chosen based on run time environment characteristics to be one of V2R6, which is Type1, T2_32Bit, which is Type 2, or T2_Linux64, which is Type 2.
43. DisableUDTImplCastForSysFuncOp	This field disables/enables implicit cast/conversion of UDT expressions passed to system operators/functions. Conversions are from UDTs to compatible predefined types. A value of TRUE disables implicit conversions. A value of FALSE (default) enables.
44. CurHashBucketSize	The number of bits currently used for the hash bucket size - 16 or 20.
45. NewHashBucketSize	The number of bits used for the hash bucket size for next reconfig or sysinit - 16 or 20.
46. MaxLoadAWT	This field defines the maximum number of AMP worker tasks (AWT) that concurrent FastLoad and MultiLoad can use altogether. The valid range is 0 .. (60% of maxampworkertasks).



DBS Control Record – General Fields (12.0)

General Fields: (these are the defaults)

1. Version	= 6	23. Export Width Table ID	= 0
2. SysInit	= TRUE	24. (Reserved for future use)	
3. DeadLockTimeOut	= 240 (seconds)	25. DBQL Options (spec. options)	= 0
4. (Reserved for future use)		26. External Authentication	= 0 (On)
5. HashFuncDBC	= 5 (Universal)	27. IdCol Batch Size	= 100000
6. (Reserved for future use)		28. LockLogger Delay Filter*	= FALSE
7. (Reserved for future use)		29. LockLogger Delay Filter Time*	= 0
8. SessionMode	= 0 (Teradata)	30. ObjectUseCountCollectRate*	= 0 minutes (Disabled)
9. LockLogger	= FALSE	31. LockLogSegmentSize	= 64 KB
10. RollbackPriority	= FALSE	32. CostProfileId	= 0
11. MaxLoadTasks	= 5	33. DBQLFlushRate	= 600 (sec.)
12. RollForwardLock	= FALSE	34. Memory Limit Per Transaction	= 2 pages
13. MaxDecimal	= 15 (18, or 38)	35. Client Reset Timeout	= 300 (sec.)
14. CenturyBreak	= 0	36. Temporary Storage Page Size	= 4K bytes
15. DateForm	= 0 (IntegerDate)	37. Spill File Path	= /var/tdrsg
16. System TimeZone Hour	= 0	38. MDS Is Enabled	= FALSE
17. System TimeZone Minute	= 0	40. EnableCostProfileTLE	= FALSE
18. RollbackRSTransaction	= FALSE	41. EnableSetCostProfile	= 0
19. RSDeadLockInterval	= 0 (240)	42. UseVirtualSysDefault	= 0
20. RoundHalfwayMagUp	= FALSE	43. DisableUDTImplCastForSysFuncOp	= FALSE
21. (Reserved for future use)	=	44. CurHashBucketSize	= 20 bits
22. Target Level Emulation	= FALSE	45. NewHashBucketSize	= 20 bits
		46. MaxLoadAWT	= 0

DBS Control Record – File System Fields

Field	Purpose
1. FreeSpacePercent	This field is used by the DBS and the File System to determine the percentage of free space to leave on cylinders during data load operations. The valid range of values is 0 .. 75. The default value is 0 (percent). A table definition (CREATE or ALTER) overrides this value.
2. MiniCylPackLowCylProd	Determines the number of free cylinders below which it will start to perform the "MiniCylPacks" operation in anticipation of their need. Setting this field to 0 indicates that "MiniCylPacks" will only be performed when no free cylinders exist. The default value is 10.
3. PermDBSize	Determines the maximum size of a Permanent Table's multi-row Data Blocks in 512-byte sectors. The valid range of values is 14 .. 255. The default value is 127 (sectors). A table definition (CREATE or ALTER) overrides this value.
4. JournalDBSize	Determines the maximum size of Transient Journal and Permanent Journal Table multi-row Data Blocks in 512-byte sectors. The valid range of values is 1 .. 127. The default value is 12 (sectors).
5. DefragLowCylProd	Determines the number of free cylinders below which it will start to perform the "Cylinder defragmentation" operation. Setting this field to 0 disables "Cylinder defragmentation". The default value is 100.
6. PermDBAllocUnit	Determines the allocation unit for Permanent Table's multi-row Data Blocks in units of 512-byte sectors. If a Permanent Table Data Block contains multiple rows, the size of the Data Block will be a multiple of the PermDBAllocUnit. The valid range of values is 1 .. 63. The default value is 1 (sector).
7. Cylinders Saved for PERM	Used to save X number of cylinders for Perm Data only – cannot be used by Spool. Range is 1 to 100; default is 10. If the number of available cylinders falls below this value, spool files will not be allocated space.
8-14. WAL parameters	Parameters to setup options for Write-Ahead Logging. (V2R6.2 feature)



DBS Control Record – File System Fields

File System Fields:

* Parameters 8-14 are new with V2R6.2

1. FreeSpacePercent	= 0%
2. MiniCylPackLowCylProd	= 10 (free cylinders)
3. PermDBSize	= 127 (sectors)
4. JournalDBSize	= 12 (sectors)
5. DefragLowCylProd	= 100 (free cylinders)
6. PermDBAllocUnit	= 1 (sectors)
7. Cylinders Saved for PERM	= 10 (cylinders)
8. DisableWALforDBs	= FALSE
9. DisableWAL	= FALSE
10. WAL Buffers	= 20 (WAL log buffers)
11. MaxSyncWALWrites	= 5 (MaxSyncWalWrites)
12. SmallDepotCylsPerPdisk	= 2 (cylinders)
13. LargeDepotCylsPerPdisk	= 1 (cylinders)
14. WAL Checkpoint Interval	= 60 (seconds)

Notes:

1. FreeSpacePercent – determines the percentage of free space to leave on cylinders during data load operations. Overridden with CREATE or ALTER Table.
2. MiniCylPackLowCylProd – threshold at which the system will perform mini-cylpacks
3. PermDBSize – sets default data block size
7. Cylinders Saved for PERM – cylinders not available to Spool

DBS Control Record – Performance Fields

Field	Purpose
1. DictionaryCacheSize	This field defines the size of the dictionary cache for each PE Vproc on the system. The valid range of values is 64 .. 1024. The default value is 1024 (kilobytes).
2. DBSCacheCtrl	Enable or disable the performance enhancements associated with Cache Control Page-Release Interface. FALSE cause old caching rules to be used. With old cache rules, data blocks added to spool tables or used in sort operations are NOT cached. With old cache rules, data blocks for user tables are always cached.
3. DBSCacheThr	Specifies the percentage value to use to calculate the cache threshold when the DBSCacheCtrl field is enabled. The valid range of values is 0 .. 100. The default value is 10.
4. MaxParseTreeSegs	This field defines the maximum number of 64KB tree segments that the parser will allocate while parsing a request. The valid range of values is 12 .. 3000. The default value is 1000.
5. ReadAhead	Enable or disable the performance enhancements associated Read-Ahead Sequential File Access Workload. The default value is TRUE.
6. StepsSegmentSize	Defines the maximum size of the plastic steps segment. Range of values is 64 – 1024 Kbytes. Default is 1024.
7. RedistBufSize	This field determines the size in units of kilobytes of hashed row redistribution buffers, subject to certain adjustments. On systems with few virtual AMP's, a larger buffer size will usually have a positive effect on performance. However, on systems with many virtual AMP's, making the buffer size too large will cause excessive memory consumption, especially if many queries involving hashed row redistribution are run concurrently. The range of valid values is 1 to 63. The default value is 4.
8. DisableSyncScan	Enables or disables the performance enhancements associated with synchronized full table scans.
9. SyncScanCacheThr	SyncScanCacheThr - This field specifies the percentage of file system (FSG) cache that the Teradata file system can assume is available for synchronized scans of tables. It does not reserve cache for this purpose. Instead, this value specifies the amount of permanent data the Teradata file system should try to retain in memory at any one time for all tables involved in synchronized scans. Whether that much memory is truly available depends on actual workload. The valid range of values is 0 .. 100, where 0 indicates that the default value should be used. The default value is 10.
10. HTMemAlloc	Specifies the percentage of memory to be allocated to a hash table for a hash join. A value of 0 turns off hash joins. Valid range is 0 to 10. The default value is 0.
11. SkewAllowance	Specifies a percentage factor used by the optimizer in deciding on the size of each hash join partition. It makes allowance for data skew in the build relation. Valid range is 20 to 80. The default value is 75.
12. Read Ahead Count	ReadAhead Count - If the ReadAhead field is set TRUE, ReadAhead Count is used to specify the number of data blocks that will be preloaded in advance of the current file position while performing sequential scans. The valid range of values is 1 .. 100. The default value is 1.
13. PPICacheThrP	This field specifies the percentage value to be used for calculating the cache threshold used in operations dealing with multiple partitions. Valid range is 0 to 500. The default is 10.
14. ReadLockOnly	ReadLockOnly – This field is used to disable or enable the special read-or-access lock protocol on DBC.AccessRights table during access rights validation and on dictionary tables accessed by read-only queries during request parsing. The default value is FALSE. FALSE enables this feature.
15. IAMaxWorkloadCache	This tunable field defines the maximum size in megabytes of the workload cache, which is used during index analysis. Valid range is 32 to 128. The default value is 32 (megabytes).
16. MaxRequestsSaved	This field governs the number of request cache entries allowed on a PE. The default number of entries that can be saved in the cache is 600. The user can modify this field to a fixed value that ranges from 300 to 2000 and, is a multiple of 10.
17. UtilityReadAheadCount	Specifies the number of data blocks the Teradata utilities will preload at a time while it performs its sequential scan. The utilities uses this field instead of the ReadAhead and ReadAhead Count fields. The valid range of values is 1 .. 100. Default is 10 blocks.
18. StandAloneReadAheadCount	Specifies the number of data blocks to preload when File System startup or a utility runs as a standalone task. The valid range of values is 1 .. 100. Default is 20 blocks.
19. DisablePeekUsing	This field enables or disables the performance enhancements associated with exposed using values. A value of F enables the feature. A value of T disables the feature. The default is F.
20. IVMaxWorkloadCache	This tunable field defines the maximum size in megabytes of the workload cache used by Index Wizard Validation. Valid range is 1 to 32. The default value is 1 (megabytes).



DBS Control Record – Performance Fields

Performance Fields:

1. DictionaryCacheSize	= 1024	
2. DBSCacheCtrl	= TRUE	
3. DBSCacheThr	= 10%	
4. MaxParseTreeSeg	= 1000	
5. ReadAhead	= TRUE	
6. StepsSegmentSize	= 1024 (kilobytes)	
7. RedistBufSize	= 4 (kilobytes)	
8. DisableSyncScan	= FALSE	
9. SyncScanCacheThr	= 10%	
10. HTMemAlloc	= 2%	
11. SkewAllowance	= 75%	
12. Read Ahead Count	= 1	
13. PPICacheThrP	= 10	(new with V2R5)
14. ReadLockOnly	= FALSE	(new with V2R5.1, FALSE enables this feature)
15. IAMaxWorkloadCache	= 32 (megabytes)	(new with V2R6.0)
16. MaxRequestsSaved	= 600 (default)	(new with V2R6.1)
17. UtilityReadAheadCount	= 10	(new with V2R6.2)
18. StandAloneReadAheadCount	= 20	(new with V2R6.2)
19. DisablePeekUsing	= FALSE	(new with 12.0)
20. IVMaxWorkloadCache	= 1 (megabytes)	(new with 12.0)

Notes:

1. DictionaryCacheSize – size of data dictionary cache for each PE – 1024 is effectively 1024 KB
4. MaxParseTreeSeg – may need to raise for more complex SQL requests
6. StepsSegmentSize – defines the maximum size of the plastic steps segment in kilobytes.

DBS Control Record – Checksum Fields

CHECKSUM LEVELS

1. System Tables This field is used to set the checksum level of system tables (data dictionaries, session information, etc.).
2. System Journal Tables This field is used to set the checksum level of system journal tables (transient journal, change tables, recovery journals).
3. System Logging Tables This field is used to set the checksum level of system logging tables (RSS tables, accounting log tables, etc.).
4. User Tables This field is used to set the checksum level of user tables (includes stored procedures, user-defined functions, join indexes, hash indexes).
5. Permanent Journal Tables This field is used to set the checksum level of permanent journal tables.
6. Temporary Tables This field is used to set the checksum level of temporary and spool tables.

Valid values for these fields are NONE (default), LOW, MEDIUM, HIGH, and ALL.

A deterministic algorithm is used to determine which words (8 bytes) are used within sectors to determine the checksum. Checksums are stored in the Cylinder Index.

NOTE: Modifying field 0 results in setting fields 1 - 6 to the specified value.

CHECKSUM LEVEL DEFINITIONS

- | | |
|-----------|---|
| NONE | Do not calculate checksums (sample 0% of the disk block to generate a checksum). This percentage cannot be modified. |
| 7. LOW | Calculate checksums by sampling a low percentage of the disk block. Default is to sample 2% (8 bytes) per disk sector. Valid range is 1 - 100%. |
| 8. MEDIUM | Calculate checksums by sampling a medium percentage of the disk block. Default is to sample 33% of the disk block. Valid range is 1 - 100% |
| 9. HIGH | Calculate checksums by sampling a high percentage of the disk block. Default is to sample 67% of the disk block. Valid range is 1 - 100% |
| ALL | Calculate checksums using the entire disk block (sample 100% of the disk block to generate a checksum). This percentage cannot be modified. |



DBS Control Record – Checksum Fields

This DBS Control utility screen is used to change system-wide checksum parameter defaults based on table type. This is part of the **Disk I/O Integrity Check** feature.

DBS Control Record – Disk I/O Integrity Fields

CHECKSUM LEVELS

- | | | |
|-----------------------------|---|------|
| 1. System Tables | = | NONE |
| 2. System Journal Tables | = | NONE |
| 3. System Logging Tables | = | NONE |
| 4. User Tables | = | NONE |
| 5. Permanent Journal Tables | = | NONE |
| 6. Temporary Tables | = | NONE |

The CHECKSUM option can also be specified at the table (object) level which overrides the system setting.

```
CREATE TABLE ...
    CHECKSUM = DEFAULT | NONE | LOW |
                MEDIUM | HIGH | ALL
```

Similar syntax for
`CREATE JOIN INDEX`
`CREATE HASH INDEX`
`ALTER TABLE`

CHECKSUM LEVEL DEFINITIONS

- | | | |
|-----------|---|---------------|
| NONE | = | 0% Sampling |
| 7. LOW | = | 2% Sampling |
| 8. MEDIUM | = | 33% Sampling |
| 9. HIGH | = | 67% Sampling |
| ALL | = | 100% Sampling |

The sampling percentages can only be specified at the system level.

When used, checksums are stored in the Cylinder Index.

Modifying DBS Control Parameters

The **modify** command is used to change DBS Control parameters. After making a modification, it is necessary to **write** the update to disk.

Example: Set the Century Break value

Another example (different than the facing page) follows. To change the Century Break value, you need to use modify and write commands of the DBSControl utility.

To change the Century Break value:

1. From the command-line prompt (or the Supervisor window of the Database Window):

dbscontrol

2. From the DBS Control window:

display general

3. Use the modify command to change flag 14:

modify general 14 = 50

4. Write changes to the GDO:

write

Note: The change does not take place until the next database restart, even though the flag shows the change right away. If you DISPLAY GENERAL, you'll see the new setting, but the setting is not really available until the system is restarted.



Modifying DBS Control Parameters

To modify a DBS Control parameter, use the **Modify** and **Write** commands.

This example changes parameter 30 to a value of 10 minutes.

To save the changes, use the **Write** command.

```
Database Window (DBW)
File Help
Supervisor dbscontrol Application 2 Application 3 Application 4 DBS I/O
46. MaxLoadAWT = 0
Enter a command, HELP, or QUIT:
modify general 30=10
The ObjectUseCountCollectRate field has been modified from 10 to 10.
NOTE: This change will become effective after the DBS Control Record
has been written.
Enter a command, HELP, or QUIT:
write
Locking the DBS Control GDO...
Updating the DBS Control GDO...
Enter a command, HELP, or QUIT:

Status: Reading

display general
modify general 30=10
write

Command:
```

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Examples of Teradata system utilities that run on the database and can be accessed through the DB Window.
 - Query Session (Qrysessn)
 - DBS Control (DBSControl)
 - Ferret Utility
- The **DBS Control** utility is used to view/modify the DBS Control Record fields which ...
 - Establish system values
 - Tune performance
 - Debug/diagnose problems

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. What are four ways that you initiate an Teradata system utility (e.g., dbscontrol)?

2. Identify the purpose of the following DBS Control utility parameters.

CenturyBreak _____

DateForm _____

DictionaryCacheSize _____

MaxLoadTasks _____

PermDBSize _____

SessionMode _____

Teradata Training

Notes

Module 54



System Restarts

After completing this module, you will be able to:

- List three different ways to restart the Teradata database.
- Use the RESTART command.
- Describe the impact of ...
 - Disk(s) failure
 - Disk array controller(s) failure
 - BYNET(s) failure
 - Node failure
 - AWS failure
 - VPROC failure
- Explain the difference between a PDE dump and a UNIX panic dump.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Types of Restarts.....	54-4
Scheduled Restart/User-initiated Restart	54-4
Unscheduled Restart/Automatic Restart	54-4
Scheduled Restarts	54-6
-f option.....	54-6
-x option	54-6
-y option	54-6
-d option	54-6
-l delay option	54-6
-P option.....	54-6
-Q option	54-6
cold option.....	54-6
coldwait option.....	54-6
Restart Teradata from DB Window	54-8
RESTART Command Options.....	54-8
Restart using the “tpareset” Command	54-10
Restart Teradata using MultiTool	54-12
PDE States.....	54-14
Unscheduled Restarts.....	54-16
Disk Failure.....	54-16
Unscheduled Restarts (cont.)	54-18
BYNET Failure	54-18
Unscheduled Restarts (cont.)	54-20
Node Failure.....	54-20
VPROC Failure	54-20
AWS Failure	54-20
TPA Reset – Crashdumps	54-22
PDE DUMP.....	54-22
Allocating Crashdumps Space	54-24
Example	54-24
TPA Dump Maintenance	54-26
UNIX Panic Dumps	54-26
Review Questions	54-28

Types of Restarts

There are two types of restarts on a Teradata Database:

- Scheduled restarts
- Unscheduled restarts

Scheduled Restart/User-initiated Restart

In a scheduled or user-initiated restart, use the RESTART command from either the DBW Supervisor window or from **vprocmanager** to restart the system.

Unscheduled Restart/Automatic Restart

In an unscheduled restart or automatic restart, the system reboots without user input.

The facing page provides examples of when you might need to perform scheduled restarts, and under what conditions you might encounter unscheduled restarts.



Types of Restarts

Scheduled Restarts

- Changing system parameters (e.g., DBS Control parameter is updated)
- Software upgrades
- Configuration changes (addition of new AMPs and/or PEs)

Unscheduled Restarts

- Power failure (e.g., 8/14/2003 – the North East U.S. and parts of Canada)
- Disasters (e.g., 8/29/2005 – Katrina hurricane; 10/22/2007 – Rancho Bernardo fires)
- Hardware failure
- Software failure
- Accidents

Restart Processes

1. Spool cylinders are returned to free cylinder list (unused cylinder pool).
2. Before logons are enabled, uncommitted work is rolled back.
 - 1st Tables are re-locked for background recovery.
 - 2nd Logons are enabled in cold start.

Scheduled Restarts

The facing page shows the windows and utilities from which you can restart the Teradata Database system, the necessary commands and available restart options. Additional information about options commonly used with **tpareset** or **restart** is provided below:

-f option

The -f option, used with the tpareset command, forces all TPA nodes to participate in the tpareset regardless of their current state, without rebooting UNIX.

-x option

The -x option allows you to shut down the Teradata Database on the entire system without shutting down the operating system. This option does not automatically restart Teradata.

-y option

The -y option automatically answers yes to the confirmation prompt.

-d option

Specifies that a DBS dump be taken before doing the restart.

-I delay option

Specifies the delay interval in seconds to wait for other nodes to join the TPA configuration. This parameter controls how long a node will wait during the BYNET configuration phase of PDE initialization for other nodes to reach that point before continuing on without them.

-P option

Requests the node to panic after the DBS dump is saved.

-Q option

Requests tpareset to run in silent mode, i.e., user is not prompted for confirmation. This should be combined with any other desired option. This will not have any effect on the -P option.

cold option

If you use the cold option in conjunction with the restart tpa or restart commands, the system does not wait for AMP vprocs to complete recovery. Instead, the system places them in offline catchup. The system will enable logons **before** recovery is complete.

coldwait option

If you use the coldwait option, the system waits for down AMP vproc recovery to complete on all vprocs and brings all AMP vprocs online. The system will enable logons **after** recovery is complete.



Scheduled Restarts

Restart Teradata with	Use this command	Options
Command-line	<code>tpareset <comment></code>	<code>-f, -x, -y -d, -l, -Q, -P</code>
DB Console - Supervisor	<code>restart tpa <comment></code>	<code>cold, coldwait</code>
vprocmanager	<code>restart</code>	<code>cold, coldwait</code>
MultiTool (Windows or Linux)	<code>reset (via GUI choices)</code>	<code>GUI menu choices</code>

Example:

```
# tpareset -f Change of system parameters
```

To see when restarts occur and brief explanation of how/why for the last week:

```
LOGON tdpid/systemfe,service;
EXEC ALLRESTARTS (DATE - 7,);
LOGOFF;
```

The “tpatrace” command may also be used to see information about restarts.

```
# tpatrace 3 (shows last 3 restarts)
```

Restart Teradata from DB Window

In a scheduled restart, you may restart the system using the RESTART command from the DBW Supervisor screen or from vprocmanager.

RESTART Command Options

The RESTART command provides the following options:

DUMP – Default is NODUMP. This option can request that the Teradata Database restarts with or without a crashdump. DUMP=YES:NO.

COLD – A full restart, but transaction recovery will be deferred. This option allows the system to determine whether a down processor is to be kept off-line, or brought back on-line while recovery is being performed. The amount of updating to be performed on the down processor is the determining factor.

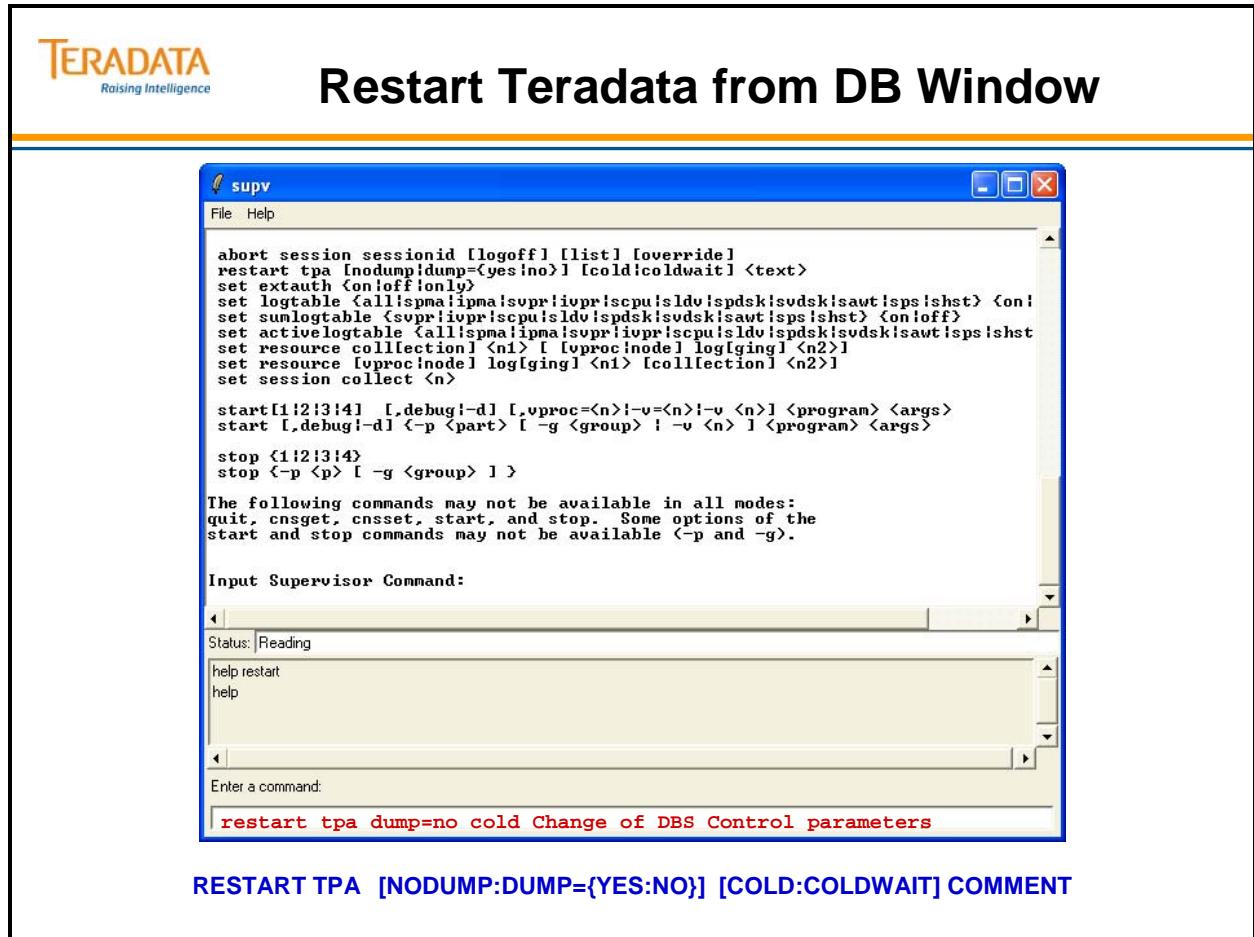
Recovery tasks are performed in the background after the system becomes available for use. Objects involved in recovery tasks are locked until the recovery is complete. All other objects are accessible to users.

COLDWAIT – A full restart, but DBS startup will be held up until transaction recovery is complete. This option specifies that all recovery options must be completed before logons are enabled. All recovered AMP vprocs are placed on-line.

COMMENT – Enter a note explaining why the restart occurred. This entry is mandatory.

You can parse the RESTART command using commas. There is no mandatory order for the keywords (dump option and the restart kind).

Note: The SET RESTART command and Set Restart Type screen set the restart type to use during the next restart of the system.



Restart using the “tpareset” Command

The **tpareset** command can be used to restart the Teradata database. Common options used with **tpareset** include:

-f option

The -f option, used with the tpareset command, forces all TPA nodes to participate in the tpareset regardless of their current state, without rebooting UNIX.

-x option

The -x option allows you to shut down the Teradata Database on the entire system without shutting down UNIX. This command shuts down Teradata without restarting Teradata.

Restart information is logged in numerous locations depending on the operating system. For example, with UNIX MP-RAS systems, the following locations are utilized.

- SW_Event_Log table – view with Software_EventLog[V] view
- Console Log (e.g., /etc/.osm)
- /var/admstreams (MP-RAS)



Restart using the “tpareset” Command

Example of using the tpareset command:

```
# tpareset -f Change of DBSControl parameters
```

```
You are about to restart the database  
on the system  
'tdt5-1'
```

```
Do you wish to continue (yes/no) [no]: yes  
tpareset: TPA reset submitted.
```

Example of using the tpatrace command:

```
# tpatrace
```

```
TPA Initialization Trace for Node 001-04
```

```
02/19/2008 15:23:59 ----- PDE starting
```

```
02/19/2008 15:23:59.51 (367) ---- PDE starting.  
02/19/2008 15:23:59.51 (367) State is NOTPA/START.
```

```
:
```

```
02/19/2008 15:24:01.19 (367) Syncing into group 6... done in 0 seconds.
```

```
:
```

```
02/19/2008 15:24:04.83 (367) State is TPA/READY.
```

```
:
```

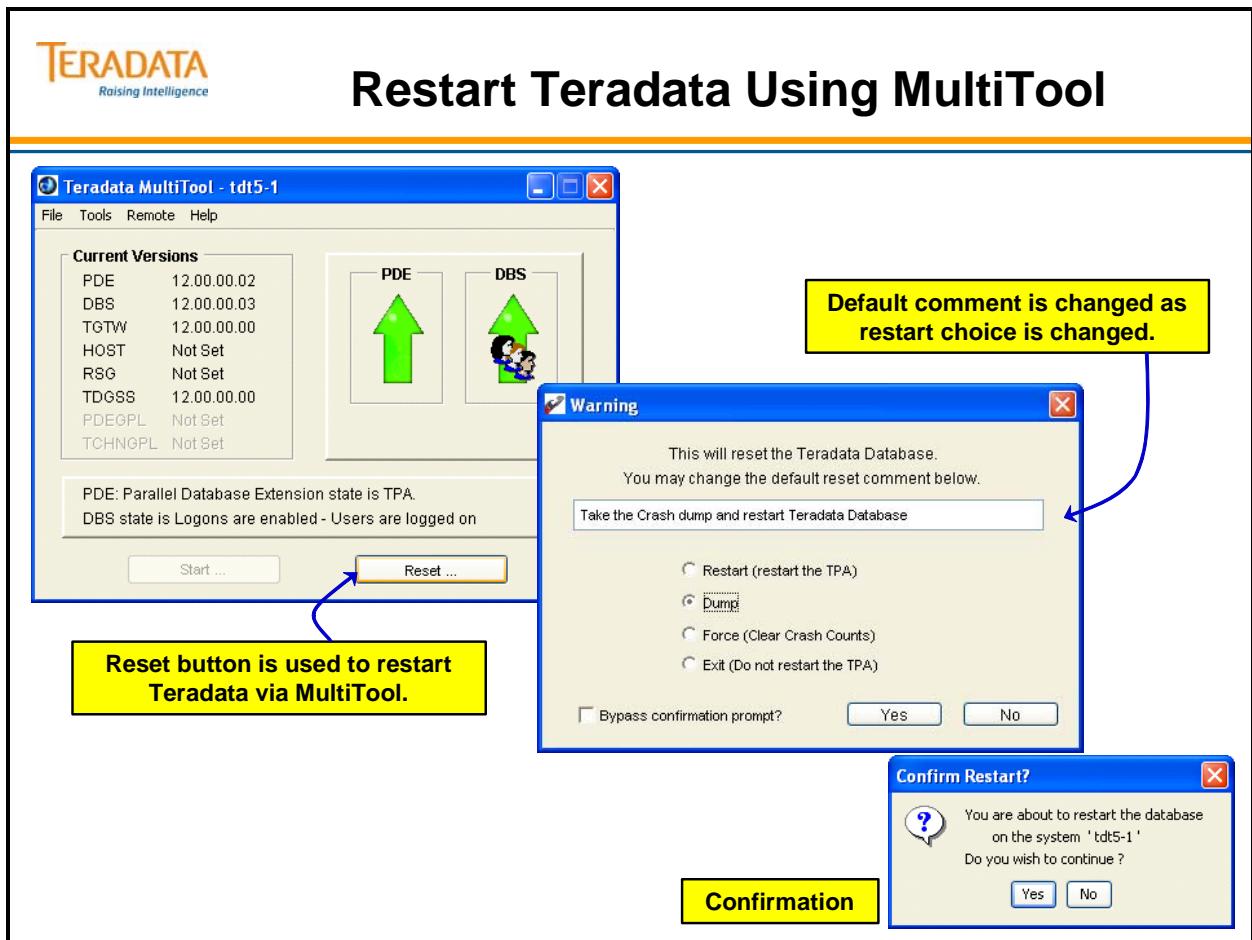
```
02/19/2008 15:24:04.86 (367) PDE started in 5 seconds.
```

Restart and status information is logged to numerous locations:

- SW_Event_Log table – view with Software_EventLog[V] view
- Console Log (e.g., /etc/.osm)
- /var/adm/streams (MP-RAS)

Restart Teradata using MultiTool

The **MultiTool** utility has a “Reset” button which can be used to restart Teradata. The facing page illustrates an example of restarting a system using MultiTool.



PDE States

The **/ntos/bin/pdestate** command can be used to check the current state of the PDE and Teradata software. PDE has three major operational states: NULL, NOTPA, and TPA. When PDE starts up, it transitions between them in that order. When PDE is shut down, it goes back to NULL state. While PDE is in transition to TPA state, it goes through several substates, corresponding to different phases of startup.

NULL/START - PDE has never started on the node(s).

NULL/STOPPED – PDE is stopped on the node(s), either explicitly or due to the start-up crash count exceeding its crash limit.

NULL/RESET – “Real” state when PDE is in reset or down state.

NULL – TPA is down on a node due to a late-joiner or other reason. The node(s) will respond to any type of reset.

NOTPA/START – the PDE is reading the local vconfig.GDO to get the TPA node list. It starts the kernel event daemons.

NOTPA/NETCONFIG – the PDE is waiting for all the node(s) to reach this state. There is a wait default of 6 min. (360 seconds) if node(s) are down. Note: Late-joiners fall out at this stage.

NOTPA/NETREADY – the PDE synchronizes the system GDO’s. It selects the control and distribution nodes.

NOTPA/RECONCILE – the PDE verifies that the correct level of PDE and TPA software is installed on all the nodes.

NOTPA – the tpastartup file is processed.

TPA/START – the PDE starts all Vprocs. It enters the final FSG initialization stage by opening the Pdisks.

TPA/VPROCS – initializes and synchronizes application GDO’s.

TPA/READY – the PDE start-up is complete. The DBS is not running.

TPA – the PDE start-up is complete. It also indicates that the DBS has started or is running.



PDE States

The **pdestate** command can be used to check the current state of the PDE and Teradata software for a specific node.

```
# /usr/ntos/bin/pdestate  
PDE: Parallel Database Extension state is TPA.
```

PDE has three major operational states:

NULL, NOTPA, and TPA

- NULL/START
- NULL/STOPPED
- NULL/RESET
- NULL
- NOTPA/START
- NOTPA/NETCONFIG
- NOTPA/NETREADY
- NOTPA/RECONCILE
- NOTPA
- TPA/START
- TPA/VPROCS
- TPA/READY
- TPA/DONE
- TPA

Unscheduled Restarts

Disk Failure

When a disk fails, there may be a loss of data. Tables with fallback protection continue to be 100% available. Tables without fallback protection will only be partially available.

An AMP will attempt 5 retries to a disk array before determining that it cannot access the array or its associated Vdisk.



Unscheduled Restarts

Disk Drive Failures

Scenario 1

Failure: One disk in a drive group

Result: No TPA reset

Resolution: Replace disk – Array Controllers automatically rebuild the disk

Scenario 2

Failure: Two disks in a drive group

Result: – TPA reset (1-5 minutes)

– AMP taken offline and marked as Fatal

– Fallback tables OK

– Non-fallback tables partially available

Resolution: – Replace the two disks

– Reformat LUNs or Volumes in the drive group

– Perform a table rebuild

– Restore non-fallback tables

Scenario 3

Failure: Two disks in 2 different drive groups associated with AMPs in the same cluster – 2 AMPs fail in a cluster

Result: Machine halts

Resolution: Restore User DBC and tables

Unscheduled Restarts (cont.)

BYNET Failure

If a BYNET fails, processing will resume on the other BYNET. Performance will be impacted.



Unscheduled Restarts (cont.)

BYNET Failures

Scenario 1

Failure: One BYNET fails

- Result:**
- No TPA reset
 - All traffic auto-switched to remaining BYNET
 - Impact on system performance

Resolution: Repair BYNET

Scenario 2

Failure: Both BYNETs fail

Result: Teradata halts and is not available

Resolution: Repair BYNETs

Unscheduled Restarts (cont.)

Node Failure

The facing page describes a node failure.

VPROC Failure

PE VPROC This type of failure produces very little impact on system performance. It reduces the maximum number of sessions that can be active at one time, however, and logons may take longer when a PE vproc is down.

AMP VPROC If a single vproc fails in one or more clusters, the system can continue servicing users with the other AMP vprocs. However, the performance level drops causing a slow down in performance and response time.

If two or more AMP vprocs fail in a single cluster, it halts the database system. All processing stops until the administrator brings at least one of the down vprocs back on-line.

AWS Failure

The facing page describes an AWS failure.



Unscheduled Restarts (cont.)

Node Failure

Scenario

- Failure:** **Node Fails (e.g., O.S. hangs, 2 power supplies fail, memory fails, etc.)**
- Result:**
- TPA restart (1 - 5 minutes) and vprocs migrate to other nodes in clique
 - Possible O.S. reboot (3 - 15 minutes)
- Resolution:**
- Repair node and reboot operating system
 - Restart Teradata to allow node to rejoin Teradata configuration

Vproc Software Failure

Scenario

- Failure:** **AMP or PE Vproc fails**
- Result:** TPA restart (1 - 5 minutes) and vprocs may be marked offline
- Resolution:** If necessary, run Scandisk, Checktable, and Rebuild utilities

AWS Failure

Scenario

- Failure:** **AWS fails**
- Result:** No restart of Teradata; AWS is not available to monitor/manage system
- Resolution:** Reboot or recover AWS

TPA Reset – Crashdumps

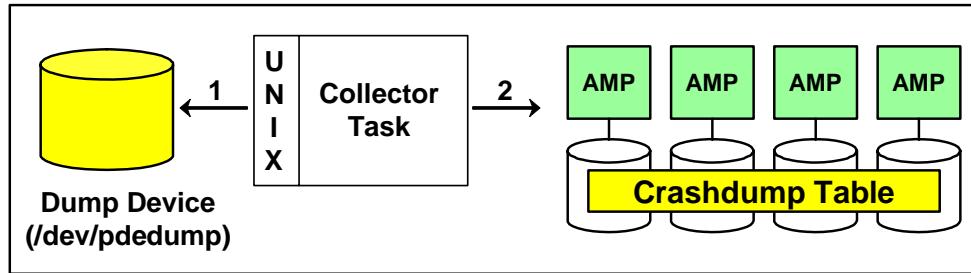
When there is an unscheduled TPA reset, a crashdump is generated (PDE dump).

PDE DUMP

A PDE dump is a selective dump of system memory; including only information that might be needed to analyze a problem within PDE or TPA (the only one currently is the Teradata Database). It can also contain pages read in from swap space that are not in memory at the time of the dump. Exact contents vary depending on the cause of the dump.

PDE dumps, being selective, vary in size depending on the system configuration and the cause of the crash. PDE dumps are much smaller than memory. PDE dumps are taken in parallel on all nodes.

TPA Reset – Crashdumps



1. Selective memory and swapped pages are written to “pdedump” space.
2. As part of Teradata restart, a background collector task reads “pdedump” and writes dump information to a Crashdump table in Crashdumps database.
 - If the Crashdumps database is out of perm space, the collector task outputs a warning message and retries every 60 minutes to create a crashdump table.

UNIX MP-RAS Commands to determine if dumps are present in “pdedump”:

```
# pdedumpcheck -v      (lists /dev/pdedump dumps that are present)
# fdlcsp - mode clear (clears all dumps from /dev/pdedump)
```

Allocating Crashdumps Space

During installation of a Teradata system, a user called CRASHDUMPS is created as a child of user DBC. Crashdumps is allocated 1GB of permanent space. Teradata recommends you allocate enough space to this database to hold three crash dumps.

Dump size is approximately 150 - 250 MB per node. Dump size can vary depending on the number of vprocs running, how busy the system was at the time of the crash, and a number of other factors. The use of fallback approximately doubles the total space requirement.

If a site needs additional space for the Crashdumps database, increase the MaxPerm space by submitting the MODIFY USER statement.

Example

The diagram on the facing page illustrates how to calculate the appropriate amount of permanent space for the Crashdumps database.

A site has a four-node system. The administrator needs to allocate enough space for three crash dumps. The formula is:

$$\begin{aligned} ((200 \times 4) \times 3) &= 2400 \text{ MB without fallback} \\ ((200 \times 4) \times 3) \times 2 &= 4800 \text{ MB with fallback} \end{aligned}$$

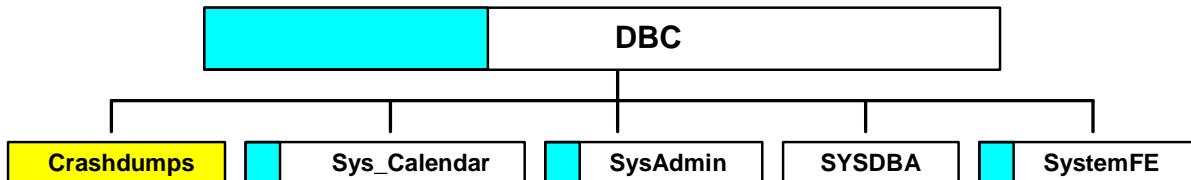
You should monitor the actual Crashdumps space usage and adjust it up or down as appropriate, depending on the size of the dumps a site typically gets.

To modify Crashdumps space, you need to log on to the system as user DBC and submit the following SQL statement:

MODIFY USER Crashdumps AS PERM = 2400E6;



Allocating Crashdumps Space



Allocate approximately 150 – 250 MB of permanent space per node per crashdump.

Example: Four-node system and you want to allocate space for three Crashdumps:

$$\begin{aligned} ((200 \times 4) \times 3) &= 2400 \text{ MB without fallback} \\ ((200 \times 4) \times 3) \times 2 &= 4800 \text{ MB with fallback} \end{aligned}$$

MODIFY USER Crashdumps AS PERM = 2400E6;

Example of Crashdump name: Crash_20080311_184642_20
 (Date) (Time) (Segment #)

Help USER Crashdumps;

Table/View/Macro name	Kind	Comment
Crash_20080311_184642_20	T	PDE:12.00.00.02,TDBMS:12.00.00.03,TGTW:12.00.00.00;

TPA Dump Maintenance

The facing page describes tasks for maintaining TPA dumps.

UNIX Panic Dumps

A UNIX panic dump is a complete dump of system memory including PDE and Kernel information, but only for the node(s) that panicked. If UNIX panics on multiple nodes, you get a separate dump for each one.

Since this is a complete dump of memory, dump size is equal to the memory size on your system.



TPA Dump Maintenance

Is the Crashdump needed?
(Contact support center
if in doubt.)

No
Yes

DELETE from Crashdumps

Optionally, delete from pdedump device

Options:

- Allow access to system via network
- Archive to file and ftp to support center
- Use DUL and archive to tape

UNIX MP-RAS Operating System Dumps

Complete dump of system memory, including:

- PDE
- Kernel

Crash utility may be used to interpret dump.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. What is the operating system command to restart Teradata? _____
2. What is the DB Window supervisor command to restart Teradata? _____
3. Which of the following choices will cause a Teradata restart? _____
 - A. AWS hard drive failure
 - B. Single drive failure in RAID 1 drive group
 - C. Two drive failures in same RAID 1 drive group
 - D. Single SMP power supply failure
 - E. SMP CPU failure
 - F. One of BYNETs fails
 - G. LAN connection to SMP is lost

Teradata Training

Notes

Module 55



Maintenance and Recovery Utilities

After completing this module, you will be able to:

- Explain how to use the following utilities to recover and maintain a Teradata database:
 - Ferret Packdisk
 - Ferret Defragment
 - Ferret Scandisk
 - Checktable
 - Table Rebuild
 - Recovery Manager
 - Showlocks
 - UpdateSpace
 - Vprocmanager
- List the order in which to execute the Checktable and Scandisk utilities.
- Name the utility and the command to initialize a Vdisk that has failed.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Maintenance, Diagnostic, and Recovery Utilities.....	55-4
Abort Host Utility	55-4
Ferret – Defragment and Packdisk.....	55-6
SCOPE Command.....	55-6
DEFragment Command.....	55-6
PACKDISK Command	55-6
Checking Data Integrity	55-8
Ferret – Scandisk Utility	55-10
Starting Scandisk.....	55-10
Stopping Scandisk.....	55-10
Checktable Utility	55-12
Starting Checktable	55-12
Stopping Checktable	55-12
Checktable – Levels of Checking	55-14
Teradata Recommendations	55-14
Checktable – Example	55-16
To Specify Indexes or Large Objects.....	55-16
Table Rebuild Utility	55-18
Recovery Manager Utility.....	55-20
Starting Rcvmanager.....	55-20
Stopping Rcvmanager	55-20
Recovery Manager Commands	55-22
Rcvmanager – List Status	55-24
ONLINE TRANSACTION RECOVERY JOURNAL.....	55-24
Rcvmanager – List Locks	55-26
Rebuild & Recover Priority	55-26
Rcvmanager – List Status (2 nd Example).....	55-28
AMP CATCHUP JOURNAL COUNTS	55-28
Rcvmanager – List Rollback Tables	55-30
Rcvmanager – Cancel Rollback on Table.....	55-32
Vprocmanager.....	55-34
Showlocks Utility.....	55-36
Report Contents.....	55-36
Orphan or Phantom Spool Issues	55-38
Update Space Utility	55-40
Summary	55-42
Review Questions	55-44

Maintenance, Diagnostic, and Recovery Utilities

Utilities available through HUTCNS and the Database Window:

Examples:

- | | |
|-------------------|--|
| Showlocks | Displays host utility (HUT) locks that are held on databases and tables during archive and restore operations. |
| Rcvmanager | Displays the number of unprocessed transactions and down-AMP-recovery journal rows, as well as locks held by transaction recovery. |

Recovery utilities available only through the Database Window:

Examples:

- | | |
|--------------------------|--|
| Ferret Packdisk | Packs cylinders together to free up cylinders |
| Ferret Defragment | Packs blocks within a cylinder |
| Ferret Scandisk | Checks disk structures per file system rules. |
| Checktable | Provides four types of internal data structure checking. Three checking levels are selectable. |
| Table Rebuild | Rebuilds data on an AMP vproc that was down, using fallback data from the other vprocs in a cluster. |
| Abort Host | Utility aborts all outstanding transactions on behalf of a channel host that is no longer operating |

Abort Host Utility

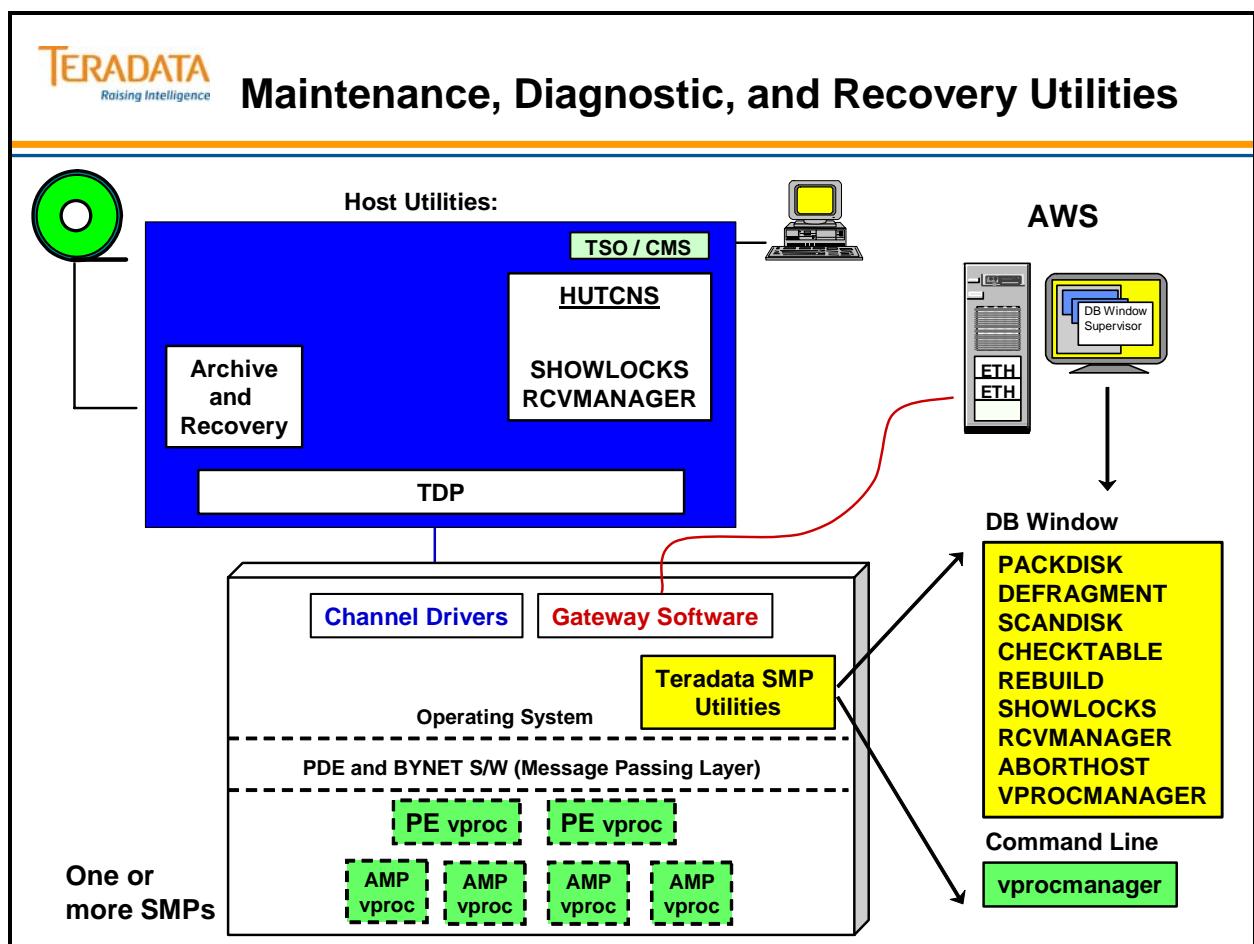
The Abort Host utility aborts all outstanding transactions on behalf of a channel host that is no longer operating. The Transient Journal will be used to rollback changes, spool files will be released, and sessions ended.

To start the Abort Host utility, enter **start aborthost** in the Supervisor interactive area.

While in the utility window, enter the following command to abort transactions that are executing from the host number or ID (nnn is the following syntax).

abort host nnn

This utility is not discussed in detail in this module.



Ferret – Defragment and Packdisk

SCOPE Command

The SCOPE command defines the range of tables and/or vprocs to display or reconfigure with the Defragment and Packdisk commands.

The facing page has an example of the SCOPE and PACKDISK commands.

DEFRAGMENT Command

Over time, it is possible that INSERTs and DELETEs can cause cylinders to become fragmented. If this is the case, the DEFFRAGMENT command may be used to defragment the cylinders on an AMP (or the system) depending on SCOPE options.

PACKDISK Command

The PACKDISK command alters a disk to reconfigure the cylinders within the scope defined by the SCOPE command. PACKDISK uses the default Free Space Percent or a new percentage specified as part of the command to pack the entire disk or a single table.

The allowable scope for PACKDISK is vprocs or tables, but not both.

The system will automatically perform mini-cylpacks when the number of cylinders falls below a certain internal threshold value. The PACKDISK command can be used to force this situation.

Starting PACKDISK

PACKDISK is a command within the Ferret utility. To start PACKDISK, enter **packdisk fsp = nnn** (where fsp = free space percent and nnn equals the percentage of cylinder free space) in the command window of the Ferret partition. Key the command in uppercase, lowercase or a combination of both. Note the interactive area where the utility has been started.

Stopping PACKDISK

To terminate the PACKDISK command, enter **ABORT**.



Ferret – Defragment and Packdisk

DEFRAGMENT

combines free sectors and moves them to the end of a cylinder.

PACKDISK

fill (or packs) cylinders up to the FSP (Free Space Percentage).

The screenshot shows a Windows-style window titled "ferret localhost". The window contains a text-based log of commands and their execution. Three blue arrows point from the left margin to specific lines in the log, highlighting the process of starting the packdisk operation.

```
ferret localhost
File Help
The SCOPE has been set
Ferret ==>
packdisk fsp=10
Fri Apr 15, 2005 06:08:14 : Packdisk will be started
On All AMP vprocs
Do you wish to continue based upon this scope?? <Y/N>
y
Fri Apr 15, 2005 06:08:20 : Packdisk has been started
On All AMP vprocs
Type 'ABORT' to stop the command before completion
Type 'INQUIRE' to check on progress of command
inquire
Inquire request has been sent
Fri Apr 15, 2005 06:08:39
Slowest vproc 0 is 20% done
Fastest vproc 5 is 21% done
The packdisk is about 21% done
Type 'ABORT' to stop the command before completion
Type 'INQUIRE' to check on progress of command
```

Status: Reading

```
packdisk fsp=10
y
inquire
```

Enter a command:

Checking Data Integrity

There are two utilities that are used to check data consistency.

- SCANDISK - checks the AMP's file system structures (CIs and DBs for consistency)
- CHECKTABLE - checks for consistency in internal data structures such as table headers, SI subtables, row identifiers, etc.

Checking Data Integrity

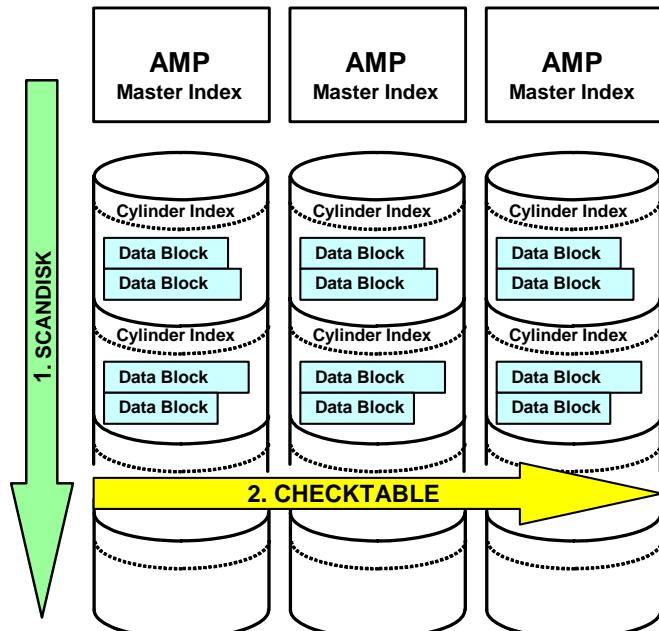
SCANDISK

Checks the AMP's file system structures (CIs and DBs for consistency)

CHECKTABLE

Checks for consistency in internal data structures such as table headers, SI subtables, row identifiers, etc.

Typically, first execute SCANDISK, then CHECKTABLE.



Ferret – Scandisk Utility

The SCANDISK utility/command enables you to determine if there is a problem with the AMP file system and assess its extent. SCANDISK is a diagnostic tool designed to check for inconsistencies between key file system data structures such as the master index, cylinder index, and data blocks.

As an administrator, you can perform this procedure as preventative maintenance to validate the file system, as part of other maintenance procedures, or when users report that there are file system problems.

Execute the SCANDISK command in the Ferret utility while the system is operational.

The SCANDISK command:

- Verifies data block content matches the data descriptor.
- Checks that all sectors are allocated to one and only one of the following:
 - Bad sector list
 - Free list
 - A data block
- Ensures that continuation bits are flagged correctly.

If Scandisk discovers a problem with a disk, you must use the Table Rebuild utility to rebuild any tables it reports as having bad data for the particular AMP vproc. (The Table Rebuild utility is discussed later in this lesson.) The output of the SCANDISK command is displayed on the screen directly after the command completes.

To avoid potential TPA resets, run SCANDISK prior to:

- Running the Checktable utility
- Rebuilding database tables using the Table Rebuild utility

Starting Scandisk

Enter **start Ferret** and from within the Ferret utility window, enter the command **Scandisk**. The SCANDISK command may be limited by the SCOPE command to scan only one table, a range of tables, or the whole vproc.

Stopping Scandisk

Scandisk terminates itself after performing the scan.

TERADATA
Raising Intelligence

Ferret – Scandisk Utility

The screenshot shows a Windows command-line interface window titled "ferret localhost". The window displays the following text:

```
Ferret ==>
scandisk
Fri Apr 15, 2005 05:45:32 : Scandisk will be started
On All AMP vprocs
Do you wish to continue based upon this scope?? (Y/N)
y
Fri Apr 15, 2005 05:45:37 : Scandisk has been started
Type 'ABORT' to stop the command before completion
Type 'INQUIRE' to check on progress of command
inquire
Inquire request has been sent
Fri Apr 15, 2005 05:46:20
SCANDISK STATUS :
Slowest vproc    0  is  28% done
Fastest vproc    6  is  29% done
The scandisk is about  28% done
Nobody has reached the SCANFREE stage
```

Below the window, there is a status bar with "Status: Reading" and a scrollable history pane containing the commands "scandisk", "y", and "inquire". At the bottom, there is an "Enter a command:" input field.

If Scandisk encounters an error for a specific AMP,
execute (table) REBUILD for that AMP.

Checktable Utility

Checktable is a diagnostic tool designed to check for inconsistencies in internal data structures such as table headers, row identifiers, and secondary indexes. Checktable can help determine if there is corruption in your system. Normally, Checktable is executed on a system that is quiescent.

Use the Checktable utility as both a diagnostic and validation tool. As a diagnostic tool, you can identify problems with data integrity. As a validation tool, you can verify data integrity prior to a reconfiguration or archive. Checktable only identifies inconsistencies; it does not correct them.

Always run Scandisk before you run Checktable. Checktable assumes the underlying structure of the file system is intact. If there are structural errors, Checktable could cause a tpa reset on the database. Scandisk is located within the Ferret utility.

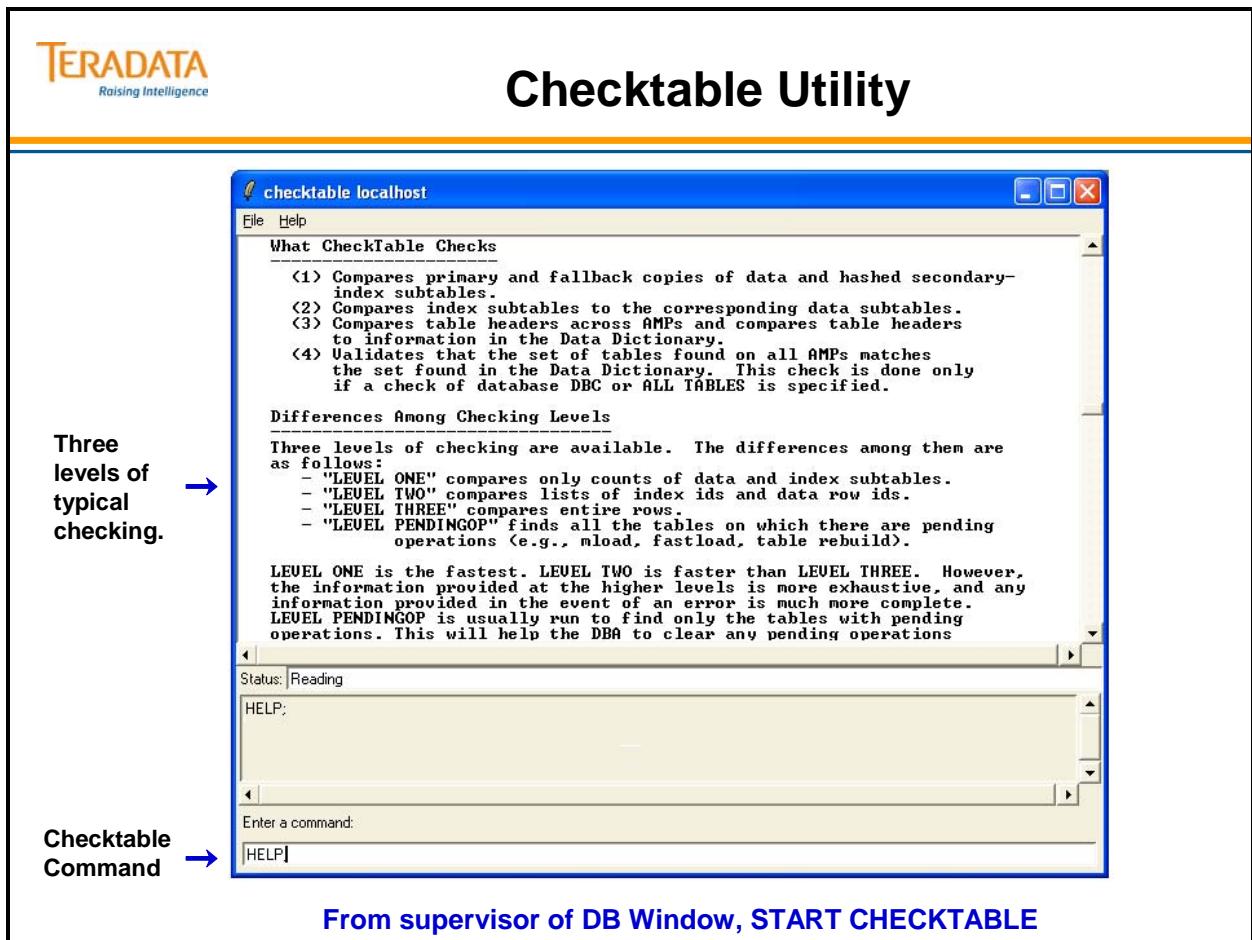
The estimated run time for a Checktable varies depending on the characteristics of the data. The more non-unique secondary indexes defined on the tables, the longer it takes to run Checktable. If you invoke Checktable when users are logged on, the time it takes to process the Checktable will depend on the activity on the system and the amount of resource contentions that it encounters (for example, object locks).

Starting Checktable

To start the utility, enter **start Checktable** from the supervisor.

Stopping Checktable

To stop Checktable, enter **QUIT**;



Checktable – Levels of Checking

The Checktable utility provides more than three levels of checking. The first three levels of checking are typically used by customers. Each level is a superset of the lower levels and runs all previous level checks.

Level One Level one checking compares the counts of data and index subtables. Use level-one checking to identify specific tables that contain errors. If errors are detected, perform a more detailed check using level-two or level-three checking.

Level Two Level two compares lists of index and row IDs as well as primary to fallback checksums. This level also verifies that hash codes reflect correct row distribution in any given subtable. Level two checking requires significantly more system resources than level one. You can use spool space to check tables that have secondary indexes.

Level Three Use level three checking to obtain detailed diagnostic information. This level provides the most detailed check and requires the most system resources. Level three compares entire rows, byte by byte.

Teradata Recommendations

Teradata recommends that you perform the following maintenance routine once a month:

1. Run a Scandisk diagnostic for all Vdisks. Scandisk performs intra-disk integrity checks by determining that the underlying file system is intact. Users may want their field support representative to start this task.
2. Follow Step 1 with a Checktable run at Level 2. The Checktable utility completes the diagnostic analysis with inter-disk integrity checks, according to the rules of the database system.



Checktable – Levels of Checking

Checktable provides three levels of checking that are typically used:

Level 1 Compares only the counts of data and index subtables.

Level 2 Includes level 1 checking; also compares lists of index and data row ids/keys and primary to fallback checksums.

Level 3 Includes levels 1 and 2 checking; also compares entire rows.

While Checktable is running, you may use the following function keys:

- F2** Displays current status.
- F3** Aborts the current table check and continues with the next.
- F4** Aborts the current Checktable command.
- F7** Help

It is recommended that you periodically (e.g., monthly) schedule the following maintenance routines:

- A SCANDISK diagnostic run for all AMPs. This function performs intra-disk integrity checks. Your Customer Engineer can start this diagnostic tool.
- A CHECKTABLE run at level 2. Checktable completes the diagnostic analysis with inter-disk integrity checks.

Checktable – Example

The facing page shows the output from executing Checktable against several databases.
Additional options in Checktable are:

Error Limit for a Check Command

The error limit is the maximum number of errors that can be found during checking a table. If the number of found errors exceeds the error limit, Checktable stops checking on the current table but continues to check the next table.

To by-pass tables which are locked.

The SKIPLOCKS option is intended to help the user by-pass any contention on tables. Without this option Checktable will block indefinitely on the table to be checked until it has been unlocked. When this option is specified Checktable will automatically skip the in-use (locked) tables.

To Check database(s) or table(s) in serial or parallel.

SERIAL/PARALLEL mode allows the user to specify whether the Checktable utility should check the specified databases/tables in SERIAL mode or in PARALLEL mode. Default mode <checkmode> is SERIAL mode.

In SERIAL mode, the Checktable utility checks a single table at a time.

In PARALLEL mode, the Checktable utility checks the specified database(s)/table(s) in parallel. The number of tables that can be checked simultaneously in parallel depends on the resource availability. A status command can be used to determine the number of parallel checks being performed at any given point of time.

To control resource consumption

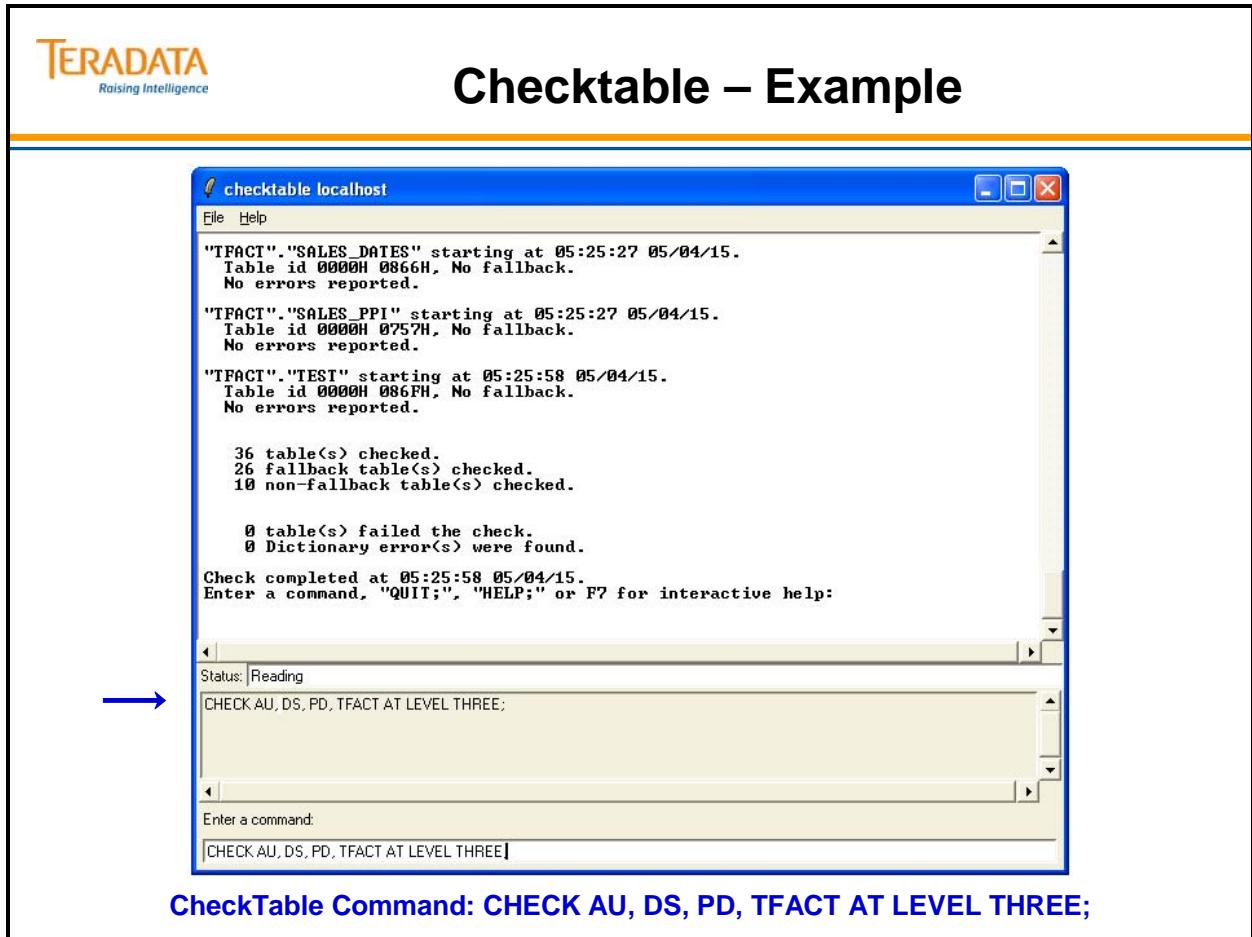
The CheckTable utility runs with MEDIUM priority by default. The user has an option to specify a performance group name with the command. Checktable will then run in the priority for the performance group name. The user can also specify LOW, MEDIUM, HIGH, or RUSH priority explicitly.

Specify the first character L (=LOW), M (=MEDIUM), H (=HIGH) or R (RUSH) when using PRIORITY option.

This option can be used to control the resource consumption by the Checktable utility.

To Specify Indexes or Large Objects

Use BUT ONLY or BUT NOT to include or exclude data subtables in the check.



The screenshot shows a window titled "checktable localhost" displaying the results of a "CHECK TABLE" command. The output indicates that three tables ("SALES_DATES", "SALES_PPI", and "TEST") were checked starting at 05:25:27 on 05/04/15. All tables had a table id of 0000H 0866H and no fallback. No errors were reported. A summary at the bottom shows 36 tables checked, 26 fallback tables checked, and 10 non-fallback tables checked. There were 0 failed tables and 0 dictionary errors found. The check completed at 05:25:58 on 05/04/15. The command prompt "Enter a command, \"QUIT;\", \"HELP;\" or F7 for interactive help:" is visible. A blue arrow points to the status bar at the bottom left of the window.

```
"TFACT"."SALES_DATES" starting at 05:25:27 05/04/15.  
Table id 0000H 0866H, No fallback.  
No errors reported.  
  
"TFACT"."SALES_PPI" starting at 05:25:27 05/04/15.  
Table id 0000H 0757H, No fallback.  
No errors reported.  
  
"TFACT"."TEST" starting at 05:25:58 05/04/15.  
Table id 0000H 086FH, No fallback.  
No errors reported.  
  
36 table(s) checked.  
26 fallback table(s) checked.  
10 non-fallback table(s) checked.  
  
0 table(s) failed the check.  
0 Dictionary error(s) were found.  
Check completed at 05:25:58 05/04/15.  
Enter a command, "QUIT;", "HELP;" or F7 for interactive help:  
  
Status: Reading  
CHECK AU, DS, PD, TFACT AT LEVEL THREE;  
  
Enter a command:  
CHECK AU, DS, PD, TFACT AT LEVEL THREE;
```

CheckTable Command: CHECK AU, DS, PD, TFACT AT LEVEL THREE;

Table Rebuild Utility

Table Rebuild is a utility that repairs data corruption. It does so by rebuilding tables on a specific AMP vproc based on data located on the other AMP vprocs in the fallback cluster.

Table Rebuild can rebuild data in the following subsets:

- The primary or fallback portion of a table
- An entire table (both primary and fallback portions)
- All tables in a database
- All tables that reside on an AMP vproc

Table Rebuild performs differently based on the type of table (e.g., fallback or not, etc.)

Type of Table	Action
Fallback tables	Delete the table header (one row table which defines a user data table) Delete specified portion of the table being rebuilt Rebuild the table header Rebuild the specified portion of the table
Non-fallback tables	Delete the table header Rebuild the table header
Permanent journal tables	Delete data for the table being rebuilt Rebuild the table header Locks the table (“pending rebuild”) Note: You must restore non-fallback data.

Note: You can start REBUILD when the failed AMP has been fixed and brought back to the OFFLINE state.

The command syntax is shown on the facing page.

TERADATA
Raising Intelligence

Table Rebuild Utility

For an AMP that has failed (e.g., a Vdisk failure) and has been repaired, REBUILD will ...

- **Rebuild table headers and data for fallback tables.**
- **Only build table headers for non-fallback tables.**

The screenshot shows the 'rebuild localhost' window. The command history pane displays the following text:

```

rebuild localhost
File Help
Release U2R.06.00.00.04 Version 06.00.00.04
TABLE REBUILD Utility <Dec 94>
  <ALL TABLES  > <ALL   >
REBUILD AMP nnnn <dbname  > <PRIMARY > DATA [, Options];
  <dbname.tname > <Fallback>

REBUILD AMP nnnn FALBACK TABLES [, Options];
RESTART REBUILD ON AMP nnnn ;
Options : LOG INTO logbase.logtbl
          NO LOCK [ON NO FALBACK TABLES]
          <DATABASE LOCK>
          WITH <TABLE  LOCK>
          <ROWRANGE LOCK>
Enter command, "QUIT ;" or F7 for help :
REBUILD AMP 1 AU ALL DATA, WITH DATABASE LOCK;
System time is 05/04/15 06:25:58
D 06:25:58 Rebuilding database AU
06:25:58 Rebuilding table Accounts
06:25:59 Complete rebuild of table Accounts
06:25:59 Rebuilding table Customer
06:25:59 Complete rebuild of table Customer
06:25:59 Rebuilding table Trans
06:25:59 Complete rebuild of table Trans
06:25:59 Complete rebuild of database AU
  
```

The status bar at the bottom left says 'Status: Reading'. The command input field contains 'REBUILD AMP 1 AU ALL DATA, WITH DATABASE LOCK;'.

Recovery Manager Utility

The Rcvmanager (Recovery Manager) utility lets you monitor the backlog out of incomplete transactions on tables that may be locked for access. The resulting journal is called the Transaction Journal. Rcvmanager also shows the count of records of data presently in the Down AMP Recovery Journal. This journal represents the data rows an AMP vproc must recover from the other AMP vprocs in the cluster before coming back online to the database.

The Recovery Manager utility runs only when the system is in one of the following states:

- Logon
- Logon/Quiet
- Logoff
- Logoff/Quiet
- Startup (if system has completed voting for transaction recovery).

If the system is not in one of the above states, Recovery Manager will terminate immediately after you start it.

Starting Rcvmanager

To start Recovery Manager, enter **start rcvmanager** in the Supervisor interactive area.

Stopping Rcvmanager

After you start Recovery Manager, you cannot stop it with the Supervisor program STOP command.

You must use the Rcvmanager **QUIT;** command to stop the program.

Note: All Rcvmanager commands end with a semicolon (;).



Recovery Manager Utility

RCVMANAGER provides a means for the user to interact with the recovery subsystem.

Following a Teradata restart, you can ...

- View the number of rows being rolled back via the Transient Journal (TJ).
- View the number of rows being updated on an AMP via the Recovery Journal (Fallback tables and rows that were updated while the AMP was out of service).
- List the tables that are locked until the rollback completes.
- Change the priority of Rollback and/or Recovery operations.

Starting with Teradata V2R5.1, you can also ...

- View which tables are being rolled back.
- Set the rollback priority for a specific session to a specific performance group.
- For a table or tables, cancel rollback processing for an online user requested abort or following a system restart.
 - WARNING: The target table will be unusable after this command is issued.
- View the tables for which rollback processing is not pending cancellation during the online transaction recovery.
 - The table is removed from the list when no more TJ rows exist for the table.

Recovery Manager Commands

The **LIST STATUS** command displays information about recovery operations in progress. The processor id option provides additional detailed information about a specific down AMP.

The **LIST LOCKS** command displays a list of all locks currently held by online transaction recovery.

The **LIST ROLLBACK TABLES** command displays the list of tables which are currently being rolled back in the system. Separate listings are generated to distinguish between online transaction recovery and system recovery. Table ids can be selected from this listing for executing the CANCEL ROLLBACK ON TABLE command. In case a '*' character follows the table names then they cannot be specified in the CANCEL ROLLBACK ON TABLE command.

The **LIST CANCEL ROLLBACK TABLES** command displays the list of tables for which rollback processing is pending cancellation during the online transaction recovery. These tables are removed from the list when all the journal rows corresponding to the tables have been skipped on all the AMPS.

The **PRIORITY** command can be used to either display or set the current priorities for rebuild or recovery. For example:

RECOVERY PRIORITY; displays the current recovery priority setting.

RECOVERY PRIORITY LOW; sets the recovery priority to Low.

DEFAULT PRIORITY; sets the recovery priority to Low and the rebuild priority to Medium.

The **CANCEL ROLLBACK ON TABLE** command is used to specify a table for which rollback processing is to be cancelled for an online user requested abort or during system recovery. The DBC password is required to execute this command. Multiple tables can be specified by separating their table ids with commas. **WARNING:** The target table will be unusable after this command is issued, and will become usable only when the table is dropped and created again, or when the table is restored from an archived backup, or when a **DELETE ALL** operation is performed on that table. The CANCEL ROLLBACK command should only be used in cases where the rollback will take longer than the restore of the table, or in cases where the table is unimportant (i.e., a temporary table). A single table retrieve operation can be performed on the target table by using the **READ OVERRIDE** locking modifier on it.

The **ROLLBACK SESSION... PERFORMANCE GROUP** command can be used to either display or set the current performance group of the rollback for a particular session. The priority associated with the specified performance group is used to change the priority of the rollback for the specified host-id and session number.



Recovery Manager Commands

From supervisor: **START RCVMANAGER**

Commands are:

LIST STATUS [<proc-id>] ; shows status of transaction and/or down AMP recovery

LIST LOCKS; displays all locks currently held by online transaction recovery

REBUILD PRIORITY [Low | Medium | High] ; sets the table rebuild priority

RECOVERY PRIORITY [Low | Medium | High] ; sets the AMP recovery priority

DEFAULT PRIORITY; sets both priorities back to their default

HELP;

QUIT;

New Commands starting in V2R5.1 include:

LIST ROLLBACK TABLES; – view the tables being rolled back for which rollback processing is not pending cancellation

LIST CANCEL ROLLBACK TABLES; view the tables that are pending cancellation

CANCEL ROLLBACK ON TABLE <table-id> [{, <table-id>} ...] ;

ROLLBACK SESSION <host>, <session> **PERFORMANCE GROUP** [<Perf Group Name>];

Rcvmanager – List Status

The Recovery Manager utility uses two basic commands: the LIST STATUS and LIST LOCKS commands. These commands display information about online transaction recovery and offline AMP recovery.

The LIST STATUS command generates two reports:

- ONLINE TRANSACTION RECOVERY JOURNAL
- DOWN AMP CATCHUP JOURNAL

ONLINE TRANSACTION RECOVERY JOURNAL

This report pertains to online transaction recovery and displays a list of all active recovery sessions as well as the maximum number of transaction journal rows remaining to be processed for the AMP that has this maximum count. Since all AMPs must complete the processing of a given recovery session before the processing of the next session begins, this information is sufficient to compute the worst-case count of transaction journal entries to be scanned.

The online transaction recovery journal counts are updated by each AMP every time a checkpoint is taken. Thus, every time an AMP checkpoints, the system will decrement its online transaction recovery journal count by 1000 and a later LIST STATUS command may display different results. If there are no recovery sessions active, then the report displays only the title.

The entries in this report are:

<i>Recovery Session</i>	ID of active recovery session
<i>Count</i>	Maximum number of transaction journal rows remaining to be processed for a specific AMP
<i>AMP W/Count</i>	The AMP to which the corresponding count applies

TERADATA
Raising Intelligence

Rcvmanager – List Status

The screenshot shows a Windows application window titled "rcvmanager localhost". The window displays two separate sessions of the "list status" command output. Each session shows the following information:

```

Release U2R.06.00.00.04 Version 06.00.00.04
RCUMANAGER Utility <Jan 96>

Enter command, "QUIT;" or "HELP;" :
list status;

ONLINE TRANSACTION RECOVERY JOURNAL COUNTS at 07:01:40 05/04/15
-----  

Recovery Session Count AMP W/Count  

-----  

1 269,584 00002

Enter command, "QUIT;" or "HELP;" :
list status;

ONLINE TRANSACTION RECOVERY JOURNAL COUNTS at 07:01:55 05/04/15
-----  

Recovery Session Count AMP W/Count  

-----  

1 266,584 00002

Enter command, "QUIT;" or "HELP;" :

```

The status bar at the bottom of the window indicates "Status: Reading".

Annotations:

- After a Teradata restart, the list status command is used.** → An arrow points from this text to the first "list status;" command in the window.
- In this example, there are almost 270K TJ rows remaining to be rolled back.** → An arrow points from this text to the first "list status;" command in the window.
- The list status command is used a second time and the count has decreased.** → An arrow points from this text to the second "list status;" command in the window.
- Note: The AMP with the highest count of TJ rows to roll back is listed.**

Rcvmanager – List Locks

The screen display on the facing page continues the same example and has an example of using the List Locks command.

The LIST LOCKS command displays all locks currently held by transaction recovery. The command generates a single report called, *Locks Held By Online Transaction Recovery*.

The report is sorted alphabetically by object name. The report does not display information for row range and row hash locks, but does display the table within which the row resides. If Recovery Manager is unable to determine the database name associated with an object, then it displays the database ID in decimal and hexadecimal. The same is true if the table name cannot be determined.

Note: LIST LOCKS displays only those locks currently held by online transaction recovery. Currently, there is no way to display locks held by offline catchup.

Rebuild & Recover Priority

The Rcvmanager utility also includes the PRIORITY command that can be used to specify priorities for:

- Table rebuild operations
- System recovery operations

Both operations are independent of each other. For either operation, if you do not explicitly set a recovery priority, the system uses the default priority. If you do not enter a new priority, the current priority setting displays. The system saves the priority settings for both operations in the Recovery Status system table.

The REBUILD PRIORITY command applies to any Table Rebuild started from the console, automatic table rebuild due to disk error recovery and MLOAD rebuild of target tables for non-participant online AMPs.

The RECOVERY PRIORITY command enables you to set a priority for the system recovery operation.

The syntax is:

```
REBUILD PRIORITY      HIGH | MEDIUM | LOW ;
RECOVERY PRIORITY    HIGH | MEDIUM | LOW ;
DEFAULT PRIORITY ;
```

Sets REBUILD PRIORITY to MEDIUM and RECOVERY PRIORITY to LOW

TERADATA
Raising Intelligence

Rcvmanager – List Locks

To determine which tables are locked as part of “Online Transaction Recovery”, the list locks command is used.

The list status command is used a third time and the count has decreased even more.

```

rcvmanager localhost
File Help
Enter command, "QUIT;" or "HELP;" :
list locks;

LOCKS HELD BY ONLINE TRANSACTION RECOVERY at 07:10:05 05/04/15
Lock      Lock          Object
Mode       Object        Name
---       ---          ---
Write     Row hash      0 1033, 0 3 (0000H 0409H, 0000H 0003H)
Write     Table         "DS"."Sales_and_SalesHistory"

Enter command, "QUIT;" or "HELP;" :
list status;

ONLINE TRANSACTION RECOVERY JOURNAL COUNTS at 07:11:55 05/04/15
Recovery   Count      AMP
Session    W/Count
1           68,584    00002

Enter command, "QUIT;" or "HELP;" :

Status: |Reading
list status;
list locks;
list status;

Enter a command:

```

Rcvmanager – List Status (2nd Example)

AMP CATCHUP JOURNAL COUNTS

This report pertains to offline AMP recovery and displays an entry for every offline AMP. Entries include the following:

AMP to be caught up	Designates which AMP needs to be recovered. Asterisk (*) indicates that the AMP would be brought into online catchup if a restart were to occur (CJ < 3000 and the OJ/TJ = 0).
AMP Status	Describes the recovery mode now in progress for the recovering AMP: NOT IN RECOVERY — AMP is still down, and has not come up into recovery yet. OFFLINE CATCHUP -- AMP is offline, and is in catchup mode ONLINE CATCHUP — The AMP is still catching up, but it is online and will accept new transactions.
Changed Row Journal Count	The number of rows updated in the cluster while an AMP was down. Usually DML changes that impact a row on a single AMP.
Ordered System Change Journal	The number of system or table level changes in the cluster. Usually DDL changes that impact row(s) on all of the AMPs.
Transient Journal Count	Number of transaction journal entries (rows) remaining in all recovery sessions on the recovering AMP: N/A - AMP is not available 0 - Local transaction recovery completed



Rcvmanager – List Status (2nd Example)

If an AMP has been offline, use the **list status** command to determine if fallback table rows are being updated from the down AMP recovery journal.

OJ (Ordered System Change Journal) – DDL changes to fallback tables.

CJ (Changed Row Journal) – DML changes to fallback tables.



```

rcvmanager localhost
File Help
Enter command, "QUIT;" or "HELP;" :
list status;

ONLINE TRANSACTION RECOVERY JOURNAL COUNTS at 07:30:59 05/04/15
Recovery Session Count AMP W/Count
----- ----- -----
DOWN AMP RECOVERY STATUS at 07:30:59 05/04/15
AMP to be caught up Pass Current Pass Next Pass
----- ----- OJ CJ OJ CJ
0004 1 0 12,236 0 64
- AMP Status: Offline Catchup
- Change Row Recovery: 32% complete in this pass

Enter command, "QUIT;" or "HELP;" :

Status: |Reading
list status;
list locks;
list status;
list status;
Enter a command:
|
```

Rcvmanager – List Rollback Tables

The **LIST ROLLBACK TABLES** command displays the list of tables which are currently being rolled back in the system. Separate listings are generated to distinguish between online transaction recovery and system recovery.

Table Ids can be selected from this listing for executing the CANCEL ROLLBACK ON TABLE command. In the situation where a '*' character follows the table names, then they cannot be specified in the CANCEL ROLLBACK ON TABLE command.

TERADATA
Raising Intelligence

Rcvmanager – List Rollback Tables

In this example, the List Rollback Tables → lists a table that is being rolled back because of a user aborted transaction.

The number of rows remaining to be rolled back and the estimated time for rollback is provided.

The Table ID and table name are also provided.

```

rcvmanager localhost
File Help
list rollback tables;

TABLES BEING ROLLED BACK AT 07:46:46 05/04/15

ONLINE USER ROLLBACK TABLE LIST
Host Session User ID Performance Group AMP W/Count
1 1001 0000:0407 R 1
TJ Rows Left TJ Rows Done Time Est.
617822 12648 00:28:52

Table ID Name
0000:060D "DS"."Sales_and_SalesHistory"

SYSTEM RECOVERY ROLLBACK TABLE LIST
Host Session TJ Row Count

Enter command, "QUIT;" or "HELP;" :

Status: |Reading
list rollback tables;

Enter a command:
|
```

Rcvmanager – Cancel Rollback on Table

The **CANCEL ROLLBACK ON TABLE** command is used to specify a table for which rollback processing is to be cancelled for an online user requested abort or during system recovery. The DBC password is required to execute this command. Multiple tables can be specified by separating their table ids with commas.

The CANCEL ROLLBACK command should only be used in cases where the rollback will take longer than the restore of the table, or in cases where the table is unimportant (i.e., a temporary table). A single table retrieve operation can be performed on the target table by using the READ OVERRIDE locking modifier on it.

WARNING: The target table will be unusable after this command is issued, and will become usable only when the table is dropped and created again, or when the table is restored from an archived backup, or when a DELETE ALL operation is performed on that table.

For example,

SELECT * FROM Sales_and_SalesHistory;

Error 7562: Invalid operation on table Sales_andSalesHistory

DELETE Sales_and_SalesHistory ALL:

Completed: 483, 350 rows processed (command executes successfully)

TERADATA
Raising Intelligence

Rcvmanager – Cancel Rollback on Table

CANCEL ROLLBACK
should only be used when

- ...
 - rollback will take longer than a restore of the table
 - or the table is unimportant (i.e., a temporary table)

You will be prompted for the DBC password.

After a cancelled rollback, the table is unusable.

SELECT * FROM ...;

- **Error 7562**

You can ...

- **DROP TABLE tbname;**
- **DELETE tbname ALL;**

The screenshot shows the rcvmanager localhost interface. The title bar says "rcvmanager localhost". The main window is titled "SYSTEM RECOVERY ROLLBACK TABLE LIST". It displays a table with columns Host, Session, TJ, Row Count, and Table ID, Name. A message in the center says "Enter command, 'QUIT;'" or "HELP;": cancel rollback on table 0000:060D;. To the right of this message is a yellow box containing the text "Table ID is provided with LIST ROLLBACK TABLES;". Below this, it says "Type the password for user DBC or press the Enter key to return: dbc". Further down, it says "Rollback will be cancelled for: 0000:060D \"DS\".\"Sales_and_SalesHistory\" Confirm y/n ? y". At the bottom, there's a status bar saying "Status: |Reading" and a command input field with the text "list rollback tables; cancel rollback on table 0000:060D; dbc y".

Vprocmanager

The Vprocmanager utility provides a means to manage/manipulate various vproc attributes. Vprocmanager can be started from the Supervisor (**start vprocmanager**) or command-line (**vprocmanager**).

Valid commands are: STATUS, INITVDISK, RESTART, BOOT, SET, HELP, and QUIT,.

STATUS

The simple format of this command (i.e., STATUS without any options) returns the DBS and PDE status tables in their entirety.

INITVDISK <VprocId>

This command initializes the DBS File System on the Virtual Disk (Vdisk) associated with the specified AMP vproc. It is only applicable to NEWPROC or FATAL AMP vprocs.

Valid VprocIds are decimal numbers in the range of 0 .. 16383. Hex number may be used with a trailing “x”.

RESTART [TPA] [NODUMP | DUMP = {YES, NO}][<RestartKind>] [<comment>]

This command is used to force different flavors of DBS restarts.

- A system dump will not be taken when the NODUMP option is specified. This is the default action.
- The DUMP = YES option causes a system dump to be taken.
- The DUMP = NO option is equivalent to NODUMP.
- Valid RestartKinds are COLD or COLDWAIT.
- The comment specifies the reason for the restart.

BOOT <VprocId>

This command will reinitialize the AMP's disk in anticipation of all-tables table rebuild and start the DBS partitions on the specified AMP. It is only applicable to vprocs with a VprocState of FATAL and a ConfigStatus of Down. A confirmation input is needed to process the initialization.

Valid VprocIds are decimal numbers in the range of 0 .. 16383. Hex number may be used with a trailing “x”.

SET

Sets the state of a vproc to either ONLINE, OFFLINE, or FATAL.

TERADATA
Raising Intelligence

Vprocmanager

Commands:

STATUS – provides status information about vprocs →

SET – sets the state (e.g., ONLINE, OFFLINE) of a vproc.

RESTART – restarts Teradata

INITVDISK – initializes the DBS File System on a Vdisk.

BOOT – initializes the File System and boots a specific vproc

The screenshot shows the Vprocmanager application window titled 'vprocmanager localhost'. The main pane displays the 'DBS LOGICAL CONFIGURATION' table with the following data:

Vproc Number	Rel. Vproc#	Node ID	Movable	Crash Count	Vproc State	Config Status	Config Type	Cluster/Host No.	RcvJrnln/Host Type
0*	1	1-01	Yes	0	ONLINE	Online	AMP	0	On
1	2	1-01	Yes	0	ONLINE	Online	AMP	0	On
2	3	1-01	Yes	0	ONLINE	Online	AMP	1	On
3	4	1-01	Yes	0	ONLINE	Online	AMP	1	On
4	5	1-01	Yes	0	ONLINE	Online	AMP	2	On
5	6	1-01	Yes	0	ONLINE	Online	AMP	2	On
6	7	1-01	Yes	0	ONLINE	Online	AMP	3	On
7	8	1-01	Yes	0	ONLINE	Online	AMP	3	On
16380	9	1-01	Yes	0	ONLINE	Online	PE	1	COP
16381	10	1-01	Yes	0	ONLINE	Online	PE	1	COP
16382	11	1-01	Yes	0	ONLINE	Online	PE	1	COP
16383	12	1-01	Yes	0	ONLINE	Online	PE	1	COP

The bottom pane shows the command history and input field:

- Status: Reading
- status;
- Enter a command:

Showlocks Utility

The Showlocks utility enables you to retrieve information about host utility locks the ARC utility places on databases and tables during backup or restore activities. This utility also displays information about locks placed during a system failure.

Host utility locks may interfere with application processing and are normally released after the utility process is complete.

If locks interfere with application processing, you can remove them by invoking the RELEASE LOCK statement. It is available through the ARC utility or as an SQL statement. An individual session may be in a "blocked" state due to one of the following situations:

- An ARC operation failed and a database cannot be accessed
- Locks were not implicitly or explicitly released after an ARC operation
- A lock was not released by the user after a system failure occurred

Showlocks can be started from DB Window Supervisor screen or HUTCNS console.

Supervisor – **start showlocks**

HUTCNS – **LOCKsdisplay** (entering LOC is all that is needed)

Report Contents

The utility displays the following information for each utility lock:

- Database name that contains lock
- Table name that contains lock (if applicable)
- User name of user who placed the utility lock
- Lock mode (read, write, exclusive, access)
- Read = Dump
- Write = Roll
- Exclusive = Restore/Copy
- Access = Group read lock or Checkpoint
- ID of vproc (all AMPs when lock resides on all AMPs)

If an object has more than one utility lock, Showlocks provides information for the most restrictive lock placed on the object.

TERADATA
Raising Intelligence

Showlocks Utility

SHOWLOCKS displays “host utility” (HUT) locks that ARC has placed on databases and tables during an archive, restore, or recovery operation.

From supervisor: START SHOWLOCKS

The ARCHIVE command places → a READ lock on database/table. →

The RESTORE/COPY command → places an EXCLUSIVE lock on database/table.

This utility displays the host utility locks that are present and terminates. There are no commands to exit this utility.

The screenshot shows a terminal window titled "Appl1 localhost" with the following text:

```

Release V2R.06.00.00.04 Version 06.00.00.04
Show Locks Utility (May 95)

This program queries all AMPs and reports all
Host Utility locks which currently exist at both
the data base level and the table level.

For each lock which is found, an entry will
appear on your console which includes the following
information:
--Data Base Name
--Table Name (if applicable)
--User Name of user who placed lock
--Lock Mode
--AMP Number

TFACT.Customer      MODE Read      AMP All Amps
TFACT.Daily_Sales   MODE Read      AMP All Amps
TFACT.Orders        MODE Excl     AMP All Amps
--ShowLocks Processing Complete--

```

The window also has sections for "Status:" and "Enter a command:".

Orphan or Phantom Spool Issues

Like all tables, spool tables require a Table ID. There is a range of tableids exclusively reserved for spool tables (C000 0001 thru FFFF FFFF) and the system cycles through them. If a spool file (table) is incorrectly not dropped, it remains in existence. Eventually, the system will cycle through all the table ids and reassign the tableid which is in use by our left over spool table. Usually, the presence of this table is detected, the query which was going to use the tableid is aborted – even though it is innocent of any wrongdoing – and the following message is returned to the user and put in the error log:

*** FAILURE 2667 Left-over spool table found: transaction aborted.

In rare cases, the leftover spool file (orphan spool) is not detected and the leftover spool is used. Since it was not created by the current transaction, its format is incorrect and the system will crash in an unpredictable way.

A more subtle condition is when a spool table is dropped, but the steps which reduce the tally of spool space currently in use are not. This is phantom spool. The tallies say the table exists, but it does not. Phantom spool does not cause restarts. Unless the space involved is a significant percentage of the total spool reserve, it is just an annoyance.

The query on the facing page can be run to flag the presence of either variety (real or phantom) of leftover spool.

Should this query return rows, the next step is to run the utility **updatespace**. This can be done while the system is in operation, but naturally; it is best done during periods of lower usage. **Updatespace** will correct the phantom spool problem. If the above query still returns rows after **updatespace** is run, then there are actual leftover spool tables. The way to get rid of them is to perform a database reset. Do not be concerned about how to avoid a restart. Do a restart and be operational faster and a whole lot surer and safer.

An important caveat about the query: notice that the user who caused the leftover spool table to be created must not be logged on when the query is run. If he/she is, then spool is considered legitimate by the query. Now, it is not uncommon for some sites to have a user which is nearly always logged on. If the leftover spool was created by such a user, this query will be not report it unless it is run when the system is quiescent.

Filer can also be used by qualified personnel in the GSC to detect real leftover spool on a quiescent system. This often has an advantage of sometimes giving important clues about the root cause.

So what do you do when you discover leftover spool? Institute a procedure to detect leftover spool tables on a regular cycle. You want to discover and eliminate them before the system attempts to reuse the id. How quickly the spool table ids are reused is very site dependent, but something near 2 weeks is the median value.



Orphan or Phantom Spool Issues

In rare cases, when a query completes (or is aborted), the spool file is not dropped.

Orphan Spool – spool file (tableid) incorrectly remains in existence

Phantom Spool – spool file tableid is gone, but the spool space remains allocated.

The following query can be executed run to determine flag the presence of leftover spool (orphan or phantom).

```
SELECT      Databasename, Vproc, CurrentSpool  
FROM        DBC.Diskspace  
WHERE       Databasename NOT IN (SELECT UserName FROM DBC.SessionInfo)  
AND         CurrentSpool > 0  
ORDER BY    1, 2  
WITH        SUM (CurrentSpool);
```

Should this query return rows, the next step is to run the utility "[updatespace](#)".

The UpdateSpace utility is also used to correct inconsistencies in the DBC.DatabaseSpace table, which might occur as the result of unusual (rare) types of system failures.

From supervisor: START UPDATESPACE

Update Space Utility

The Update Space utility (**updatespace**) recalculates the permanent, temporary, or spool space used by either of the following:

- A single database and its individual tables
- All databases in a system and their individual tables

The Update Space utility accomplishes this by performing the following:

- Examining storage descriptors and adding up space for each table.
- Setting values in CurrentPermSpace, CurrentTempSpace, or CurrentSpoolSpace in the DBC.DatabaseSpace table for each table and for the containing database as a whole.

A different utility, Update DBC (**updatedbc**), recalculates the maximum allowed values for permanent, temporary, and spool space in the DBC.Dbase and DBC.DatabaseSpace tables.

The following table lists the difference between the Update DBC and Update Space utilities.

- Update DBC recalculates maximum allowed values for permanent, temporary, and spool space.
- Update Space recalculates current usage for permanent, temporary, and spool space.

The only reason to use Update Space is to correct inconsistencies in the DBC.DatabaseSpace table, which might occur as the result of rare types of system failures.

The format of the command to use with the “Update Space” utility is:





Update Space Utility

The Update Space utility, `updatespace`, recalculates the permanent, temporary, or spool space in the `DBC.DatabaseSpace` table.

The screenshot shows a Windows application window titled "updatespace localhost". The window has a menu bar with "File" and "Help". The main area displays the following text:

```
Release U2R.06.01.00.26 Version 06.01.00.26
UPDATE SPACE Utility <Dec 94>

The Update Space program provides for recalculating the
permanent database space or temporary space or spool
space used by a single database or by all databases in
the system.

The format of the input command is:
UPDATE [SPOOL | TEMPORARY | ALL] SPACE FOR <ALL DATABASES | dbname> ;

Enter either ALL DATABASES or the name of the database
for which space is to be recalculated. The SPOOL
parameter is only allowed with a single database.

Enter Command
update all space for all databases;
Updating space for DBC
Space updated for DBC
```

Status: Reading

update all space for all databases;

Enter a command:

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- **Packdisk** – fill (packs) cylinders up to the Free Space Percentage (FSP) with the purpose of freeing up cylinders.
- **Defragment** – combines free sectors and moves them to the end of a cylinder.
- **Scandisk** – identifies and determines the extent of any problems with the AMP file system.
- **Checktable** – checks for inconsistencies in internal structures such as table headers, row identifiers and secondary indexes.
- **Table Rebuild** – repairs data corruption.
- **Recovery Manager (RCVManager)** – enables you to view information about online transaction recovery and AMP recovery.
- **Showlocks** – displays host utility locks.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. **True or False.** The Checktable utility features two levels of internal table checking.
2. **True or False.** The Table Rebuild utility rebuilds tables differently depending on whether the table is a fallback, non-fallback or permanent journal table.
3. The _____ utility does a consistency check within an AMP's file system.
4. The _____ utility does a consistency check for a table across all AMPs.
5. The _____ utility can be used to set an offline AMP to online.

Teradata Training

Notes

Module 56



Permanent Journals

After completing this module, you will be able to:

- **Describe journaling options and the type of recovery each option provides.**
- **Determine when to use permanent journals instead of (or in addition to) fallback to provide data integrity.**
- **Create, modify and delete permanent journals for databases and tables.**
- **Use the DBC.Journals view to associate tables with specific permanent journals.**

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Automatic Data Protection Mechanisms (Review).....	56-4
Transient Journal.....	56-4
Fallback Protection.....	56-4
Down AMP Recovery Journal	56-4
RAID 1 and RAID 5	56-4
Cliques	56-4
Permanent Journals	56-6
Location of Change Images	56-8
AMP Definitions	56-8
After-Image Journals Save Storage Space	56-8
Assigning Tables to a Permanent Journal	56-10
Journaling Functions	56-10
Creating a Permanent Journal	56-12
Deleting a Permanent Journal	56-12
Syntax.....	56-12
Assigning a Permanent Journal.....	56-14
Rules and Limitations	56-14
DBC.Tables.....	56-14
Syntax.....	56-14
Before-Image Journals	56-16
Before Images	56-16
After-Image Journals.....	56-18
Site Disaster	56-18
Journal Subtables	56-20
Current Journal.....	56-20
Restored Journal.....	56-20
Permanent Journal Statements	56-22
Backing up tables on a Teradata System.....	56-22
Recovery with Permanent Journals.....	56-24
Journals[x] View	56-26
Columns Defined	56-26
Summary	56-28
Review Questions	56-30

Automatic Data Protection Mechanisms (Review)

The Teradata system offers a variety of methods to protect data. Some methods are automatically activated when particular events occur in the system. Other data protection methods require that you set options. Each data protection technique offers different types of advantages under different circumstances.

Transient Journal

The Transient journal maintains snapshots of rows in tables before you or other users make changes to them. If the transaction fails or if you abort the request, the Transient Journal copies its snapshot into the existing table which rolls back any changes the failed transaction may have made to the table.

Fallback Protection

Fallback protection is an optional data protection feature that you activate with the CREATE or MODIFY commands. Fallback provides data level protection by automatically creating a copy of each row on a fallback AMP. If the primary AMP fails, the system can access the fallback copy. The fallback feature allows automatic recovery using the Down AMP Recovery Journal once the down AMP comes back on-line. Fallback protected tables occupy twice the space in your system as non-fallback tables.

Down AMP Recovery Journal

The Down AMP Recovery Journal supports fallback protection. If a primary AMP fails, the fallback feature allows automatic data recovery using the Down AMP Recovery Journal. This feature consists of these two journals: DBC.ChangedRowJournal and DBC.OrdSysChngTable.

RAID 1 and RAID 5

RAID 1 provides data redundancy through disk mirroring which means that data on one disk is identical to the information on another disk. If one disk fails, the alternate disk takes over. RAID 5 (or RAID S) protects data with a technique called “data parity protection”. Data is striped across multiple disks while the parity of each piece of data is preserved to allow array controllers to determine what the missing data is. The user experiences no downtime

Cliques

Group of SMP nodes sharing a common set of disk arrays. If a node fails, vprocs can migrate to other nodes within the clique. Although Teradata will restart, allows Teradata to continue running in the event of a node failure.



Automatic Data Protection Mechanisms (Review)

- **Transient Journal**
 - Takes before-image (snapshot) of row before change is made
 - Copies before-image of row back to table if transaction fails
- **Fallback Protection**
 - Optional data protection feature for a table
 - Creates copy of each row on fallback AMP
- **Down AMP Recovery Journal**
 - Automatically used for fallback tables when an AMP is down
 - Other AMPs in the cluster identify rows that have changed for a down AMP
- **RAID 1 or RAID 5**
 - Data redundancy through disk mirroring (RAID 1) or data parity protection (RAID 5)
 - Provides protection from physical disk failure
- **Cliques**
 - Group of nodes where vproc migration can occur
 - Provides protection from node failure(s)

Permanent Journals

The Teradata system offers a manual method called permanent journals that you can use to protect data. 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, rollback, or abort transactions. A permanent journal can capture a snapshot of rows before a change, after a change, or both. Each database or user space can contain only one journal table.

Existing data tables can write to a journal table defined in its parent or to a journal table located in another database or user. Journal tables require permanent space.

You can create permanent journal tables with the CREATE USER/CREATE DATABASE statement or the MODIFY USER/MODIFY DATABASE statement.

Permanent journal tables exist within a database or user space. Only one permanent journal can be assigned to that user or database. The journal may be located in the same database or user as the tables that use the journal or in a different database.



Permanent Journals

Permanent journals:

- Optional features that can provide protection for software and hardware failures.
- Store committed, uncommitted and aborted changes.
- Users manage journal tables.

Permanent journal options:

- Single before change image: **BEFORE**
 - Captures images before a change is made
 - Protects against software failures
 - Allows rollback to a checkpoint
- Single after-change image: **AFTER**
 - Captures images after a change is made
 - Protects against hardware failures
 - Allows rollforward to a checkpoint
- Dual image: **DUAL BEFORE** or **DUAL AFTER**
 - Maintains two images copies
 - Protects against loss of journals
- Keyword **JOURNAL** with no other keywords capture single before and after images.

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 chart on the facing page illustrates the location of change-image journals. The placement of permanent journals depends on: requested image type (either before or after) and the protection type (either fallback or non-fallback).

AMP Definitions

The following definitions are used to describe how AMPs are used to store before and/or after images.

Primary AMP	Holds before- and/or after-images for any table with fallback protection. Holds single before images and dual after-images for non-fallback protected tables.
Fallback AMP	Contains before- and/or after-images for tables with fallback protection. The DBC distributes duplicate data rows to fallback processors by assigning the row's hash code to a different AMP in the cluster.
Backup AMP	Holds three types of images for non-fallback tables: 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.

After-Image Journals Save Storage Space

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.



Location of Change Images

The location of journal rows depends on the image type requested (before or after) and the protection type of the journaled tables.

Fallback Tables

- | <u>Journal Option</u> | <u>Change Image Location</u> |
|-----------------------|------------------------------|
| After images | Primary AMP and fallback AMP |
| Before images | Primary AMP and fallback AMP |
- Dual images are always maintained for fallback tables.
 - To determine the fallback AMP for a journal row, the fallback hash maps are used.

Non-fallback Tables

- | <u>Journal Option</u> | <u>Change Image Location</u> |
|-----------------------|------------------------------|
| After images | Backup AMP |
| Before images | Primary AMP |
| Dual after images | Backup AMP and primary AMP |
| Dual before images | Primary AMP and backup AMP |
- For no fallback tables, you may request either single or dual journal images.
 - A backup AMP is another AMP in the same cluster as the primary AMP assigned to journal rows. These rows are not distributed using hash maps, but are directed to a specifically assigned backup AMP.

Assigning Tables to a Permanent Journal

When you create a new journal table, there are options you can use to control the type of information the table captures.

A permanent journal provides four basic 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 or backup AMP.
Before Image	Captures/stores row values before a change occurs.
After Image	Captures/stores row values after a change occurs.

Unlike transient and recovery journals, permanent journal options capture and store all changes whether, committed, uncommitted, or aborted. In addition, journal maintenance and activity are under user control.

Journaling Functions

Journal tables use rollback operations for software failure recoveries. To restore data tables to the state they were in before a software failure; configure a permanent journal to capture before-change information.

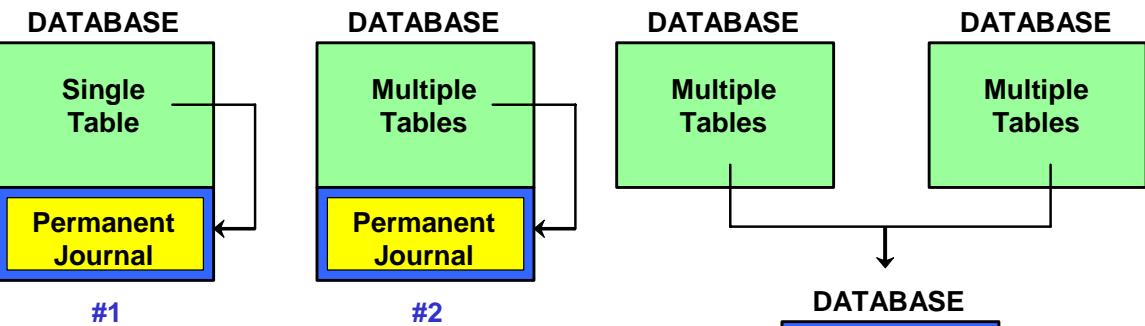
Permanent journaling is not a substitute for RAID technology or fallback protection. Both options provide duplicate images of all rows in a table. The journal tables only maintain images for changed rows.

Journal tables can protect 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 AMP vprocs fail in the same cluster. (This would mean the loss of two disks in a rank per failed AMP vproc.)
- 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 (Dual journal).



Assigning Tables to a Permanent Journal



No database or user may contain more than one permanent journal table.

The journal table may be located:

- In the same database [#1, #2 and #4] or
- In a different database than the data tables [#3 & #4]

Creating a Permanent Journal

You create permanent journals when you create a user or database. To create permanent journals within an existing user or database, use the MODIFY statement. The facing page shows examples of using these statements.

The following restrictions apply to the use of permanent journals:

- If a journal table in another user/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.
- Permanent journals 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.
- Permanent journals are not supported across certain Data Definition (DDL) statements. Statements that may prevent a rollforward or rollback operation from passing that point in the journal include:
 - ALTER TABLE
 - RENAME TABLE
 - MODIFY USER or MODIFY DATABASE
 - COMMENT

Deleting a Permanent Journal

Use the MODIFY USER or MODIFY DATABASE statement to delete a permanent journal. Before you delete the journal, you must use the ALTER TABLE statement to stop the journaling being done to that journal.

Syntax

```
ALTER [TABLE NAME]
    ,WITH [JOURNAL TABLE=JOURNAL TABLE NAME];
    ,NO BEFORE JOURNAL
    ,NO AFTER JOURNAL;

MODIFY DATABASE [DATABASE NAME AS]
    DROP DEFAULT JOURNAL TABLE=[JOURNAL TABLE NAME];
```



Creating a Permanent Journal

You create permanent journals at the user/database level when you define a new user or database:

```
CREATE DATABASE Payroll_Tab AS PERM = 100e6  
DEFAULT JOURNAL TABLE = Payroll_Jrnl;
```

Or you can create them in an existing user or database:

```
MODIFY DATABASE HR_Tab AS  
DEFAULT JOURNAL TABLE = HR_Jrnl;
```

They are identified in the DD/D as TableKind 'J':

```
SELECT DatabaseName, TableName, TableKind  
FROM DBC.Tables  
WHERE TableKind = 'J' ;
```

Response:

<u>DatabaseName</u>	<u>TableName</u>	<u>TableKind</u>
Payroll_Tab	Payroll_Jrnl	J
HR_Tab	HR_Jrnl	J

Assigning a Permanent Journal

Permanent journals are optional. You can specify journal options at the database/user level or at the individual table level. The journal options you can define are:

JOURNAL	DUAL AFTER JOURNAL
BEFORE JOURNAL	NO JOURNAL
AFTER JOURNAL	NO AFTER JOURNAL
DUAL JOURNAL	NO BEFORE JOURNAL
DUAL BEFORE JOURNAL	

You can define a DEFAULT JOURNAL TABLE associated with a user or database. You can associate an individual table within the database with the DEFAULT JOURNAL (by default) or another journal table by specifying that on the CREATE or ALTER TABLE statement.

Users activate permanent journaling by including the JOURNAL option in the CREATE or MODIFY statements for users or databases. The following page illustrates CREATE USER and CREATE TABLE statements that create and assign permanent journals.

If you create a database/user and specify a default journal table, but do not specify any journaling options, the default at the database level is NO BEFORE and NO AFTER journaling for tables created in the database. When creating a table in this database/user and if you want journaling, you must specify the journaling options you want as part of the CREATE TABLE.

Rules and Limitations

You must allocate sufficient permanent space to a database or user that will contain permanent journals. If a database or user that contains a permanent journal runs out of space, all table updates that write to that journal abort.

DBC.Tables

The DBC.Tables view can display the names of existing journal tables. The TableKind field displays the letter J for any table set up as a permanent journal. The query statement on the next page displays a list of journal table names.

Syntax

```
[NO] [BEFORE] [[,][NO] [AFTER JOURNAL]]
[DUAL] [AFTER] [[,][DUAL ] [BEFORE JOURNAL]]
DEFAULT JOURNAL TABLE = [dbname.] tname
```



Assigning a Permanent Journal

```
{ CREATE USER | CREATE DATABASE }...
[ [ NO | DUAL ] [ AFTER | BEFORE ] JOURNAL ]...
[ DEFAULT JOURNAL TABLE = journal_name ];
```

Default journal values at the database levels are:

<u>Journal Option</u>	<u>Default</u>
NONE SPECIFIED	NO JOURNAL MAINTAINED
NEITHER AFTER NOR BEFORE	BOTH TYPES IMPLIED
NEITHER DUAL NOR NO	FALLBACK – DUAL IMAGES; NO Fallback – SINGLE IMAGES

At the table level, you can indicate journal options with the CREATE statement:

```
CREATE TABLE ...
[ [ NO | DUAL ] [ AFTER | BEFORE ] JOURNAL ] ...
[ WITH JOURNAL TABLE = journal_name ];
```

Default journal values at the table levels are :

<u>Journal Option</u>	<u>Default</u>
NONE SPECIFIED	Defaults to USER/DATABASE
AFTER IMAGE ONLY	Defaults FOR BEFORE IMAGE
BEFORE IMAGE ONLY	Defaults FOR AFTER IMAGE
NEITHER DUAL NOR NO	Defaults to PROTECTION TYPE

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

Before-Image Journals

You can define permanent journals to record:

After Change Images

The data in a row after a change has occurred is recorded in the permanent journal.

Before Change Images

The data in a row prior to its change is recorded in the permanent journal.

Both

(Before Change Images
and After Change
Images)

The data in the permanent journal is maintained in the internal DBC format and is not accessible to the user through any SQL statements. Users delete permanent journals.

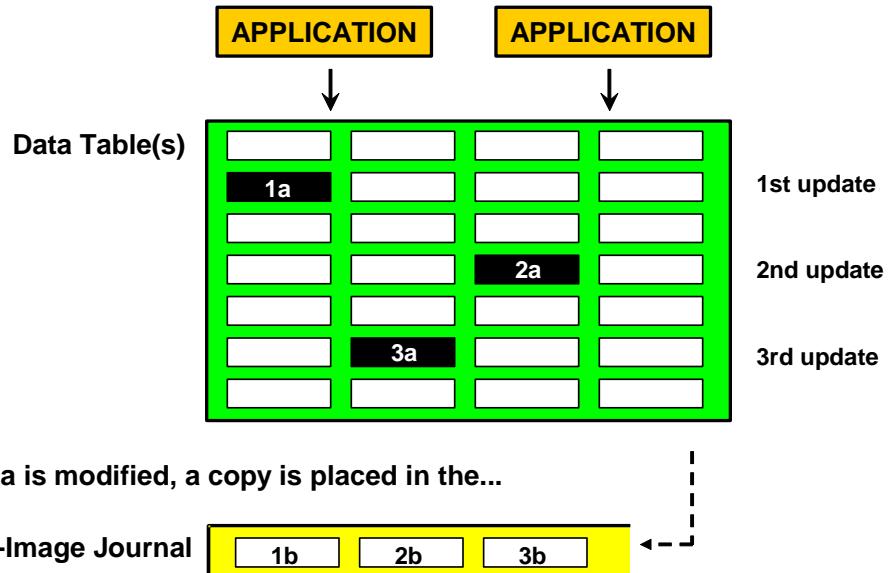
Before Images

Before Images are used for ROLLBACK recovery as shown on the following page. 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. Permanent journals roll back all transactions from a table to a checkpoint. They may not be used to roll back specific transactions.



Before-Image Journals

Before-images are used to roll back users' changes to one or more tables by returning the data to a previous consistent state.



After-Image Journals

After you create an after-image journal, a snapshot of a row value is stored in the permanent journal after a change is committed. If a hardware failure occurs, the after-image journal can roll forward any changes made to data tables since the last full system backup.

Site Disaster

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 magnetic tape daily and store the tape off-site.

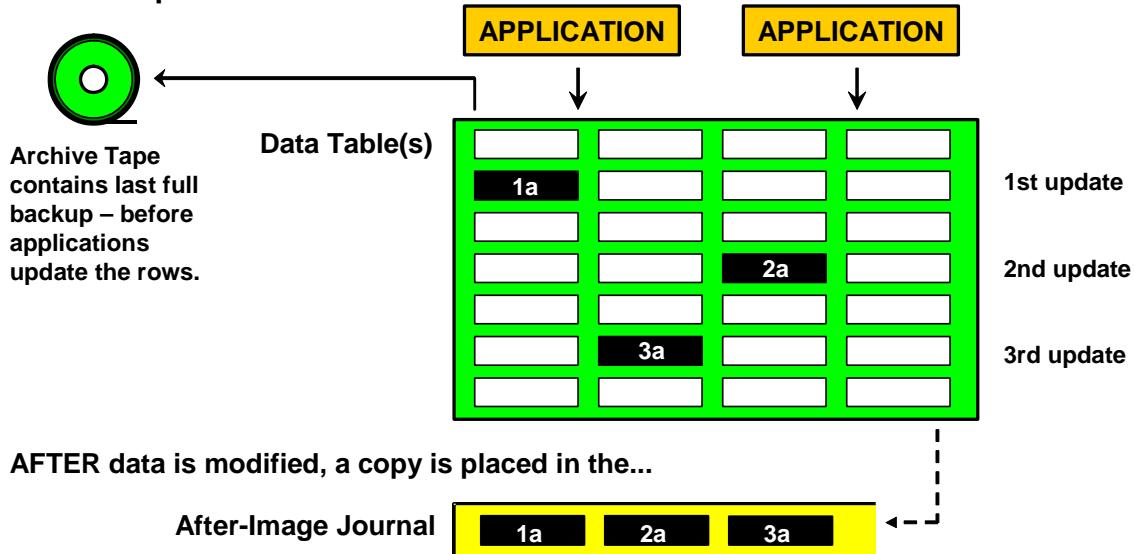
Daily archives may not be practical for very large databases. To solve this problem, you can activate after-change journals and take a daily archive of the journal itself that provides archived copies of all changes made since the last full database archive. The full backup tapes along with the journal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

The facing page shows after images in the permanent journal are used for ROLLFORWARD recovery.



After-Image Journals

After-images can be used to apply changes users have made since the last full backup.



To recover data that must be restored, use the after-images in the permanent journal to rollforward users' changes since the restored backup was taken.

Journal Subtables

Each journal table consists of three areas:

- Active area (part of current journal subtable)
- Saved area (part of current journal subtable)
- Restored area (part of restored journal subtable)

The active and saved areas together are referred to as the Current Journal. The restored subtable is called the Restored Journal. The contents and purpose of each subtable are discussed below:

Current Journal

Each time you update a data 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, you must submit the Checkpoint With Save statement. 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 statement 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, it initiates a new active subtable automatically. You can now submit an ARCHIVE JOURNAL TABLE statement. Archiving the journal saves it to tape.

Restored Journal

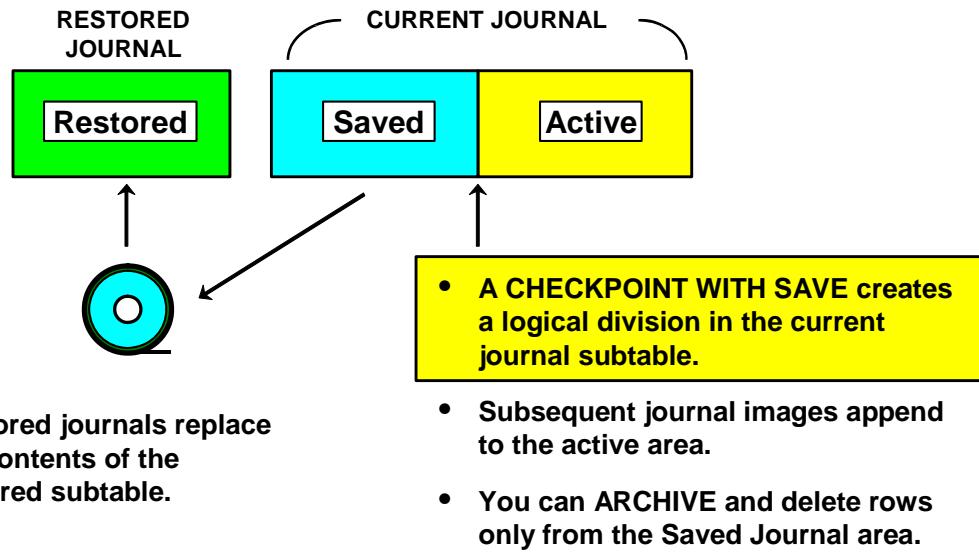
To restore a journal, move the journal table contents from the portable storage media back to the restored subtable. The information stays there until you invoke roll operations.

Permanent journals are maintained in an internal Teradata database format. They are not accessible by SQL statements and cannot be used for audit trail purposes.

Journal Subtables

A permanent journal table consists of three areas:

- **Active area** – part of Current journal
- **Saved area** – part of Current journal
- **Restored area** – part of Restored journal



Permanent Journal Statements

Use the ARC (Archive and Recovery) utility to perform backup and recovery functions associated with permanent journals. The archive and recovery functions include:

ROLLFORWARD	Replaces a data row by its after-image from the beginning of the journal, to either a checkpoint or to 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 a Teradata System

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

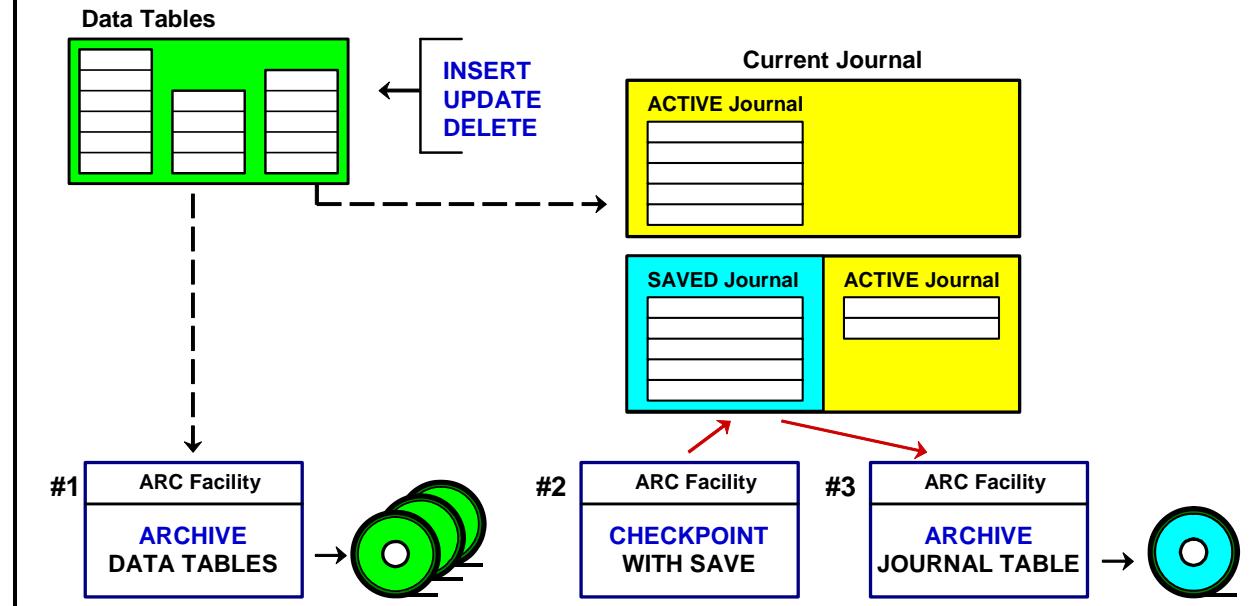
Permanent Journal Statements

1. First, ARCHIVE data tables.

After users and/or applications have modified tables, save the journal images.

2. The CHECKPOINT WITH SAVE command creates a saved journal.

3. You can ARCHIVE and delete saved journal rows.



Recovery with Permanent Journals

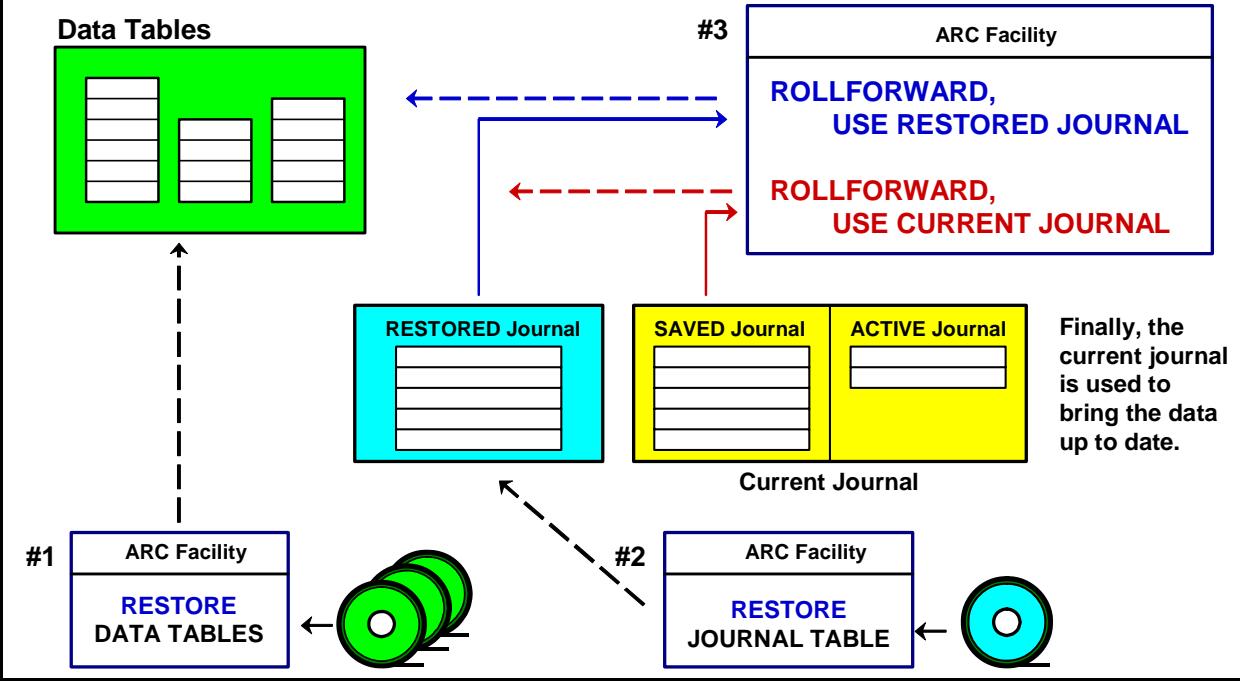
An example of how to use some of these ARC statements is shown when a batch program is run:

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. Later changes are also backed out.
3. The data table is now in its *original* condition.

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

Recovery with Permanent Journals

Tables or databases are restored first. Next, archived journals are restored, one at a time, and then the restored journal is rolled forward.



Journals[x] View

The Teradata system provides a system view called DBC.Journals, that displays links between journal tables and the data tables that journal into them. The DBC.JournalsX View is a restricted view. The restricted version of the view displays only those objects that you own or to which you hold access rights.

The example on the next page uses the SELECT statement to list all of the tables in the system that uses a permanent journal. In addition, it requests to see a list of the journal names.

The response displays the table names first followed by the journal names.

Columns Defined

The Journals view has four different columns. Each one is described below:

Tables_DB	Displays the name of a database where a data table resides that has the journal option activated.
TableName	Displays the name of a data table that records changed images in a journal table.
Journals_DB	Displays the name of a database where a journal table resides.
JournalName	Displays the name of a journal table associated with a listed data table.



Journals[X] View

Associates journals with tables when the user owns or holds rights on the objects referenced.

DBC.Journals[x]

Tables_DB	TableName	Journals_DB	JournalName
-----------	-----------	-------------	-------------

Example: List all tables in the system that use a journal and list the names of the journals.

```
SELECT      TRIM (Tables_DB)      || '.' || TableName
           AS "Table Name"          (CHAR(30))
           ,TRIM (Journals_DB)    || '.' || JournalName
           AS "Assigned to Journal" (CHAR(30))
FROM        DBC.Journals
ORDER BY    1 ;
```

Example Results:

Table Name	Assigned to Journal
HR_Tab.Employee	HR_Tab.HR_Jrn1
HR_Tab.Department	HR_Tab.HR_Jrn1
HR_Tab.Job	HR_Tab.HR.Jrn1
Payroll_Tab.Paycheck	Payroll_Tab.Payroll_Jrn1

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Permanent journals maintain a sequential history of all changes made to the rows of one or more tables.
- You create a permanent journal when you CREATE/MODIFY a user/database.
- Permanent journal image options:
 - Single before-change images
 - Capture images before a change is made and allows rollback to a checkpoint. Protects against software failures.
 - Single after-change images
 - Capture images after a change is made and allows rollforward to a checkpoint. Protects against hardware failures.
 - Dual images
 - Maintain two copies of before or after images. Protects against loss of journals.
- Use ARC facility to perform backup and recovery operations associated with permanent journals.
- The Journals[X] view provides information about links between journal tables and the tables that journal to them.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

- 1. True or False.** A permanent journal stores committed, uncommitted, and aborted changes to a row in a table.
- 2. True or False.** A database or user can have many permanent journals.
- 3. True or False.** Separate Permanent Journals are required for before and after images.
- 4. True or False.** The Saved and Active areas are both part of the Current Journal.
- 5. True or False.** The CREATE JOURNAL statement is used to create a permanent journal.
- 6. True or False.** Tables that use a Permanent Journal must be in the same database as the Permanent Journal.

Teradata Training

Notes

Module 57



A Tale of Three Tables

After completing this module, you will be able to:

- Analyze the efficiency of backup procedures after a single disk failure.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Permanent Journal Scenario.....	57-4
Table X.....	57-6
Table Y.....	57-8
Table Z	57-10
Permanent Journals	57-12
Archive Policy.....	57-14
Daily Archive Procedures	57-14
Weekly Archive Procedure	57-14
Archive Scenario.....	57-16
After Restart Processing Completes.....	57-18
After REBUILD and Restart of Teradata.....	57-20
Table X Recovery	57-22
Table Y Recovery	57-24
Recovery Action	57-24
Table Z Recovery	57-26
Recovery Action	57-26
After Recovery	57-28
Summary	57-30

Permanent Journal Scenario

In the following scenario, assume that a user has three tables in a four AMP system. Each table has its own data protection features stored on all four AMPs. The illustrations on the following pages illustrate data protection features in effect for each table.



Permanent Journal Scenario

A Tale of Three Tables

A user has three data tables:

Table X Fallback
Before and
After Image Journals

Table Y No Fallback
No Before and
Dual After Image Journals

Table Z No Fallback
Single Before and
Single After Image Journals

Table X

Table X is defined as having fallback, dual before images, and dual after images.



Table X

- ✓ Fallback protected data
- ✓ Fallback protected before images
- ✓ Fallback protected after images

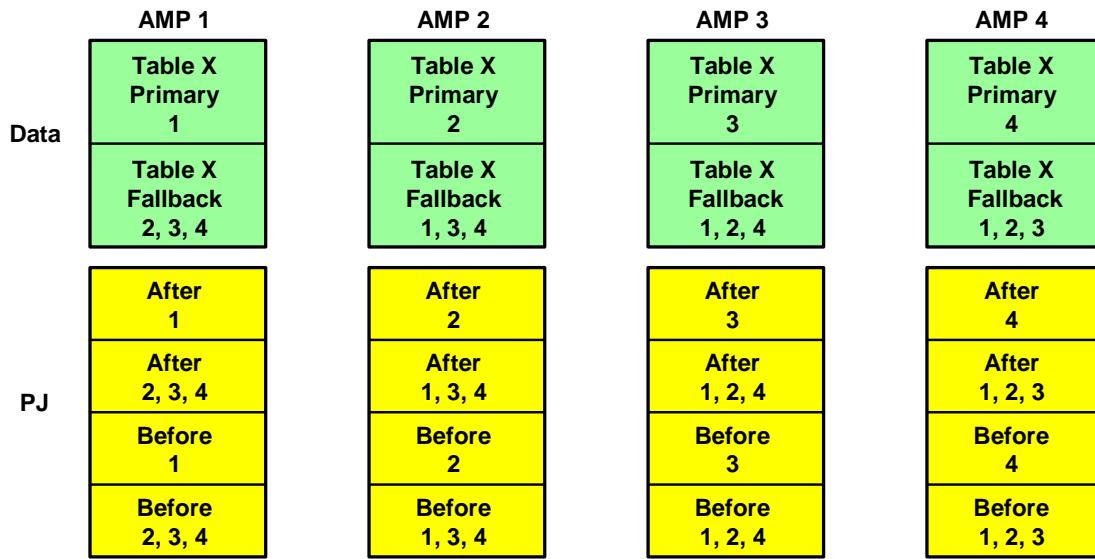


Table Y

Table Y has no fallback protection, but has dual after image journaling defined.



Table Y

- ✓ No fallback
- ✓ Dual after image

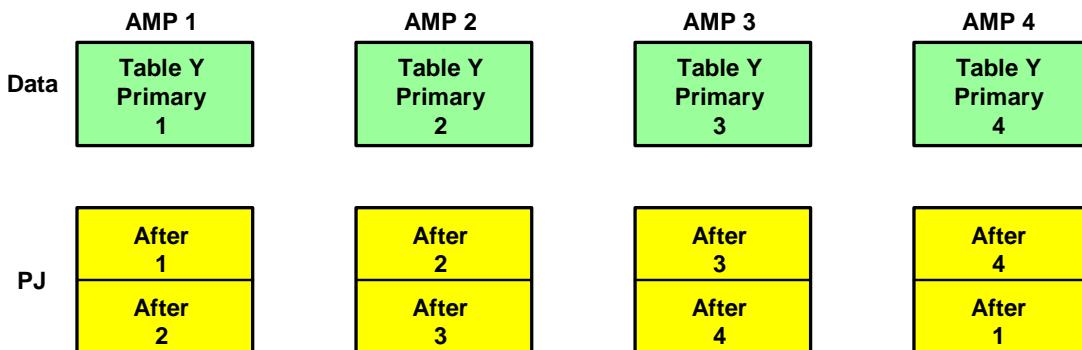


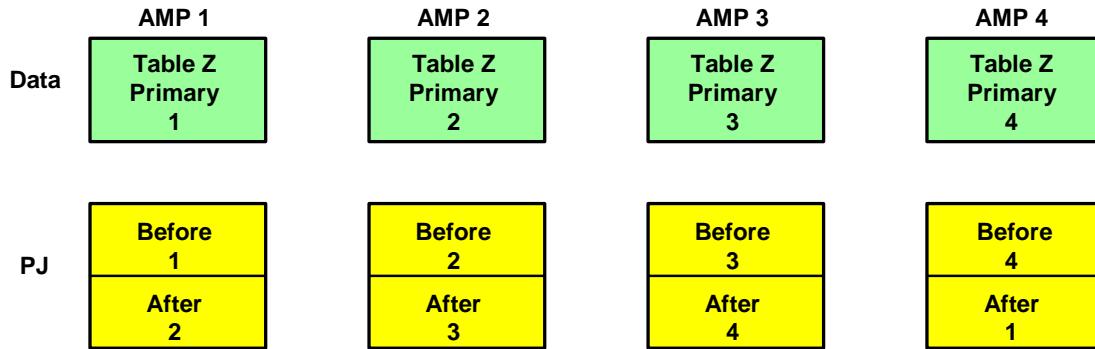
Table Z

Table Z has no fallback protection. This table has single before and after-images.



Table Z

- ✓ No fallback
- ✓ Single before images
- ✓ Single after images

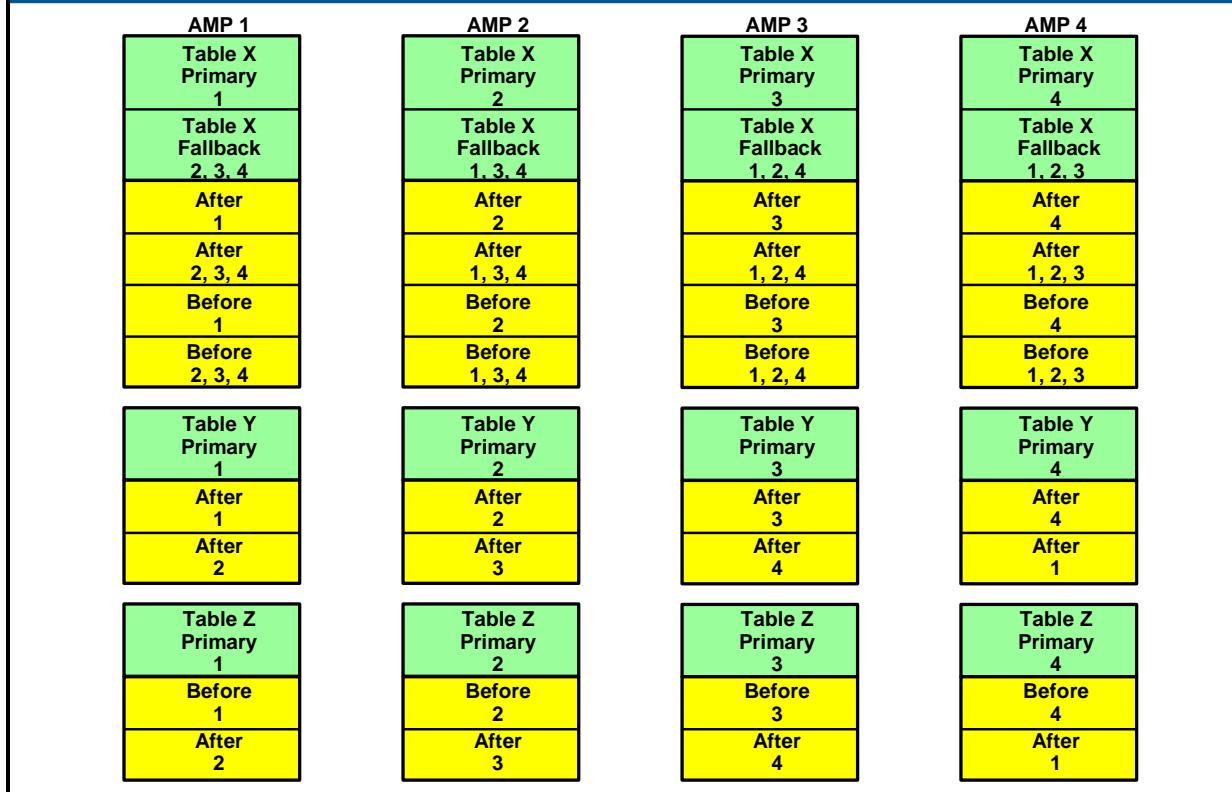


Permanent Journals

The facing page shows the three tables and all their data protection options.

Permanent Journals

(Putting all tables three together)



Archive Policy

The company established an archive policy to cover any data loss in the event of a site disaster. The archive policy has two components:

- Daily archive procedures
- Weekly archive procedures

Daily Archive Procedures

Each day the administrator submits a CHECKPOINT WITH SAVE command for each journal table which appends any changes stored in the active journal subtable to the saved journal subtable. In addition, it initiates a new active journal subtable. Second, the administrator archives each current journal, and then deletes the saved journal subtable from the saved journal.

Only one table is archived each day. By the end of the week, each table has been archived once.

Weekly Archive Procedure

Each week the administrator submits an all-AMPs ARCHIVE of all data tables.



Archive Policy



DAILY

CHECKPOINT ALL JOURNALS with SAVE
ARCHIVE JOURNAL TABLES
DELETE SAVED JOURNALS
ARCHIVE One Complete Table per Day

WEEKLY

Perform ALL-AMPS ARCHIVE of DATA TABLES

Archive Scenario

The company activated its archive policy and implemented daily and weekly backup procedures as scheduled. Each day the administrator archives journals X, Y, and Z.

On Monday, the administrator archived data table X, and on Tuesday archived table Y. On Wednesday, the administrator archived data table Z. On Thursday, two drives failed in a drive group.



Archive Scenario

Monday:

Archive journals X, Y and Z
Archive table X

Tuesday:

Archive journals X, Y and Z
Archive table Y

Wednesday:

Archive journals X, Y and Z
Archive table Z

Thursday:

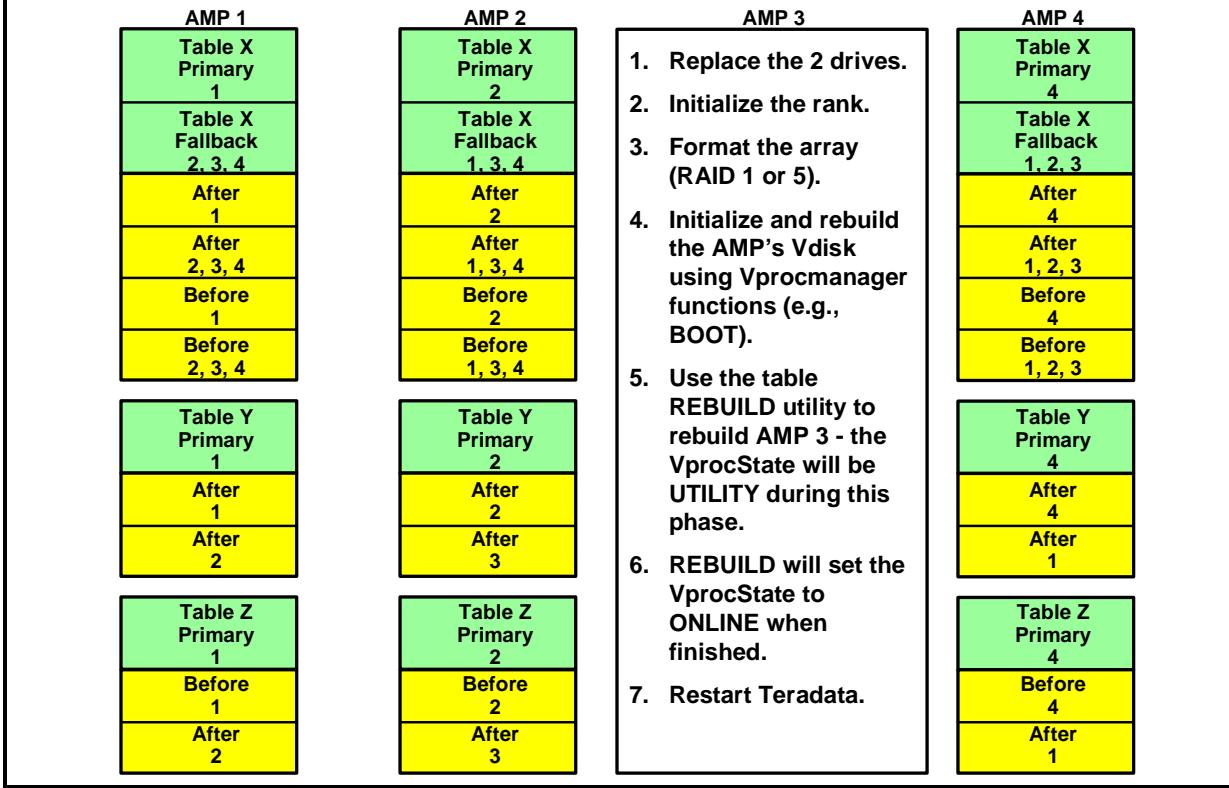
AMP 3: Two drives fail in a drive group

After Restart Processing Completes

The administrator utilized restart procedures to replace the down AMP. The diagram on the facing page outlines each restart step. Each restart procedure is explained below:

1. Replace the 2 drives.
2. Initialize the rank.
3. Format the array (RAID 1 or 5).
4. Initialize and rebuild the AMP's Vdisk using Vprocmanager functions (e.g., BOOT).
5. Use the table REBUILD utility to rebuild AMP 3 - the VprocState will be UTILITY during this phase.
6. REBUILD will set the VprocState to ONLINE when finished.
7. Restart Teradata.

After Restart Processing Completes



After REBUILD and Restart of Teradata

The diagram on the facing page shows the row information that the administrator recovered after she executed the REBUILD and RESTART commands.

Table X is fully recovered. All primary and fallback rows are restored. In addition, all before and after-journal images are recovered as well. The administrator needs to perform additional recovery measures on table Y and table Z.



After REBUILD and Restart of Teradat

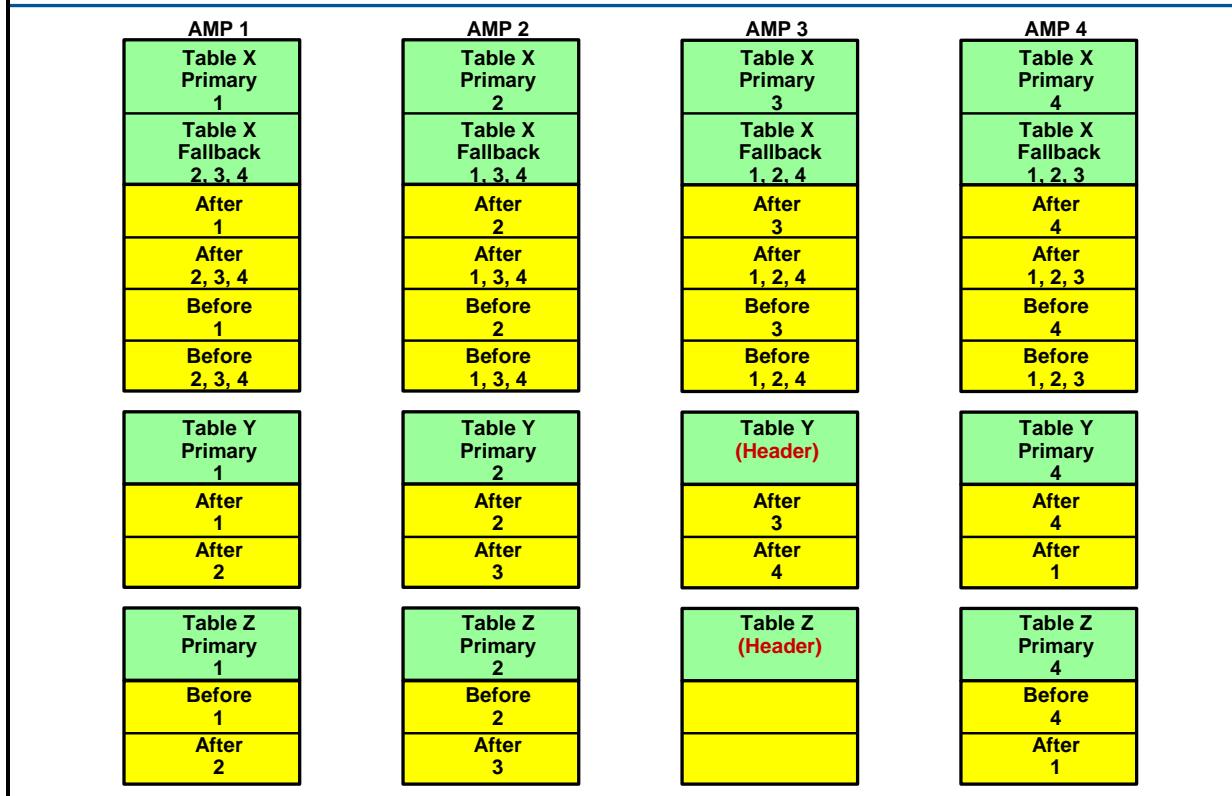


Table X Recovery

Table X has primary and fallback rows restored. All journal images are also recovered.



Table X Recovery

- ✓ Fallback protected data
- ✓ Fallback protected before images
- ✓ Fallback protected after images

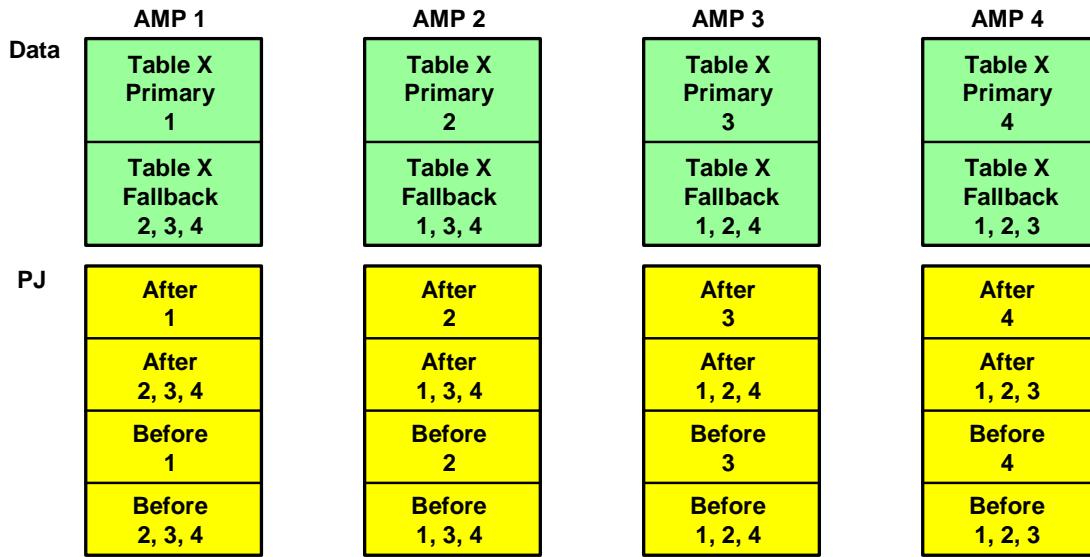


Table Y Recovery

The diagram on the facing page illustrates table Y after REBUILD and RESTART procedures.

The administrator used tables stored on AMP 2 and AMP 4 to restore the two permanent journal tables stored on AMP. The primary table is still missing. The administrator needs to perform some interactive recovery procedures to fully recover missing data for table Y on AMP 3.

The administrator will be unsuccessful if he/she attempts to access the row information from table Y. The following message may appear in response to an attempted SQL statement:

2642 AMP Down: The request against non-fallback Table_Y cannot be done.

Recovery Action

The administrator must perform the following steps to fully recover table Y:

1. Perform a single-AMP RESTORE of AMP 3 using Tuesday's ARCHIVE of table Y to restore all data rows stored in the archive file from table Y.
2. Do NOT release the utility locks.
3. Restore Wednesday's ARCHIVE of journal Y for AMP 3.
4. Perform a single-AMP ROLLFORWARD on AMP 3 using the RESTORED journal from table Y. Doing so replaces the existing rows in table Y with any after-change images made since the last backup on Tuesday.
5. Use the DELETE JOURNAL command to delete restored journal Y. This action deletes all stored images from the restored journal.
6. Perform a single-AMP ROLLFORWARD on AMP 3 using the CURRENT journal from table Y. This step replaces existing table rows with any after-change images stored in the active and/or saved subtables of the permanent journal.
7. RELEASE all utility locks.

Table Y is now fully recovered. All its contents are now available to users.



Table Y Recovery

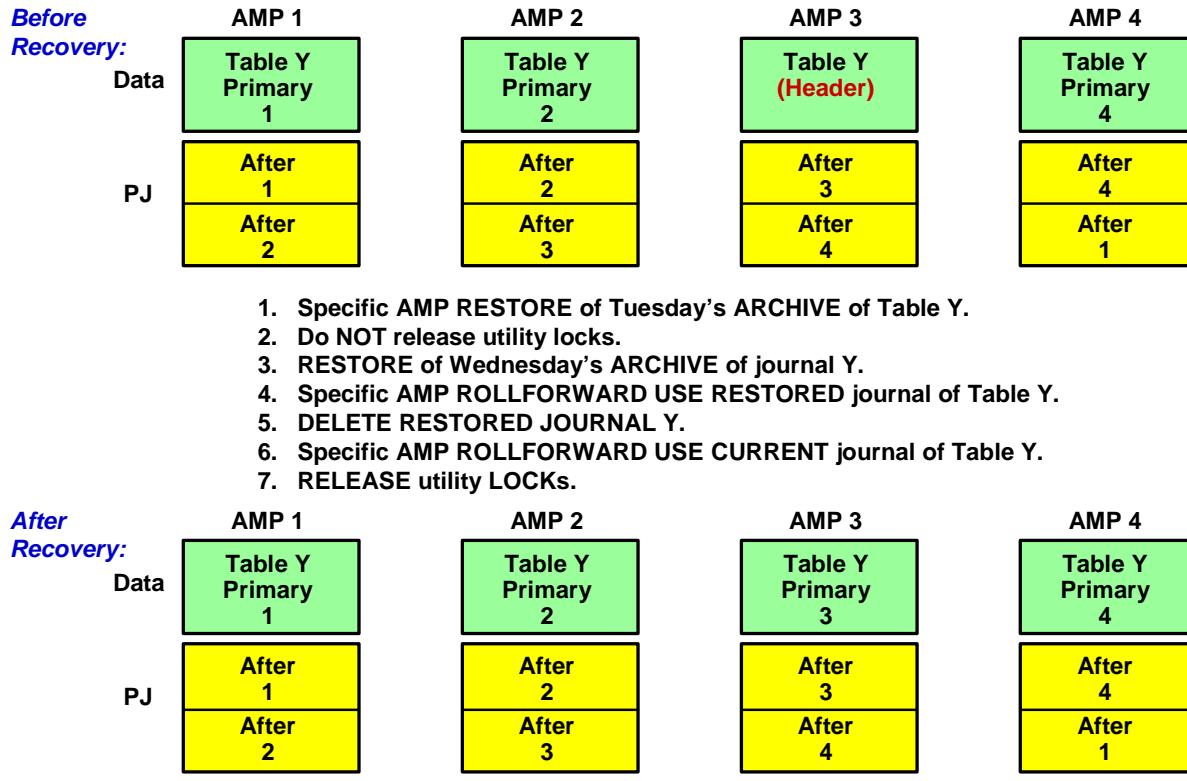


Table Z Recovery

The first diagram on the facing page illustrates table Z after REBUILD and RESTART procedures.

Neither permanent journal tables stored on AMP 3 were restored. In addition, the primary table information is still missing. The administrator needs to perform some interactive recovery procedures to fully recover the missing data for table Z on AMP 3.

Recovery Action

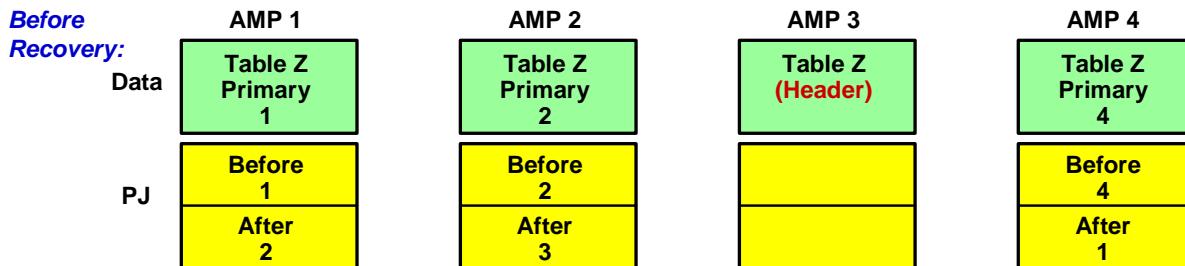
The administrator must perform the following steps to fully recover table Z:

1. Perform a single-AMP RESTORE of AMP 3 using Wednesday's ARCHIVE of table Z to restore all data rows stored in the archive file from table Z.
The administrator does not restore the journal tables for table Z since a complete backup of the table was performed on the same day as the journal archive. All changes through Wednesday would be in the archive of the entire table.
2. The administrator does NOT release the utility locks.
3. Perform a single-AMP ROLLFORWARD on AMP 3 using the CURRENT journal from table Z. This action replaces existing table rows with any after-change images stored in the active and/or saved subtables of the permanent journal. Any changes in the current journal would have occurred on Thursday before the disk failure.
4. Perform an all-AMPs archive of table Z to protect against a second disk failure in the same cluster. The administrator is unable to restore the journal for AMP 3 because she did not elect dual images. Another disk failure in this cluster leaves data unrecoverable. To correct this, the administrator deletes the saved journal and starts a new journal.
5. Perform a CHECKPOINT WITH SAVE and DELETE SAVED JOURNAL. The CHECKPOINT step moves any stored images from the active subtable to the saved subtable of the current journal and initiates the active subtable. The DELETE step erases the contents of the saved subtable since they are no longer needed.
6. RELEASE all utility locks.

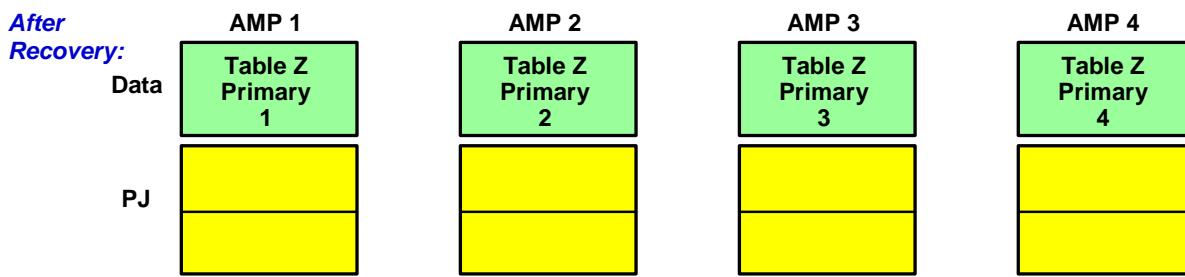
Table Z is now fully recovered. All its contents are now available to users. Notice that the table is recovered but the journals are not.



Table Z Recovery



1. Specific AMP RESTORE of Wednesday's ARCHIVE of Table Z.
2. Do NOT release utility locks.
3. Specific AMP ROLLFORWARD USE CURRENT journal of Table Z.
4. Perform all-AMPs ARCHIVE of Table Z.
5. Run CHECKPOINT WITH SAVE and DELETE SAVED JOURNAL.
6. RELEASE utility LOCKs.



After Recovery

The diagram on the facing page shows the three tables after recovery. The following summary outlines the effects of permanent journals on recovery from a single disk failure.

Fallback Tables, Dual Image Tables (Table X)

- Processing continues
- Journals play no part in recovery

No Fallback Tables, Dual Image Journals (Table Y)

- Limited processing continues
- Data and journal tables are fully recovered

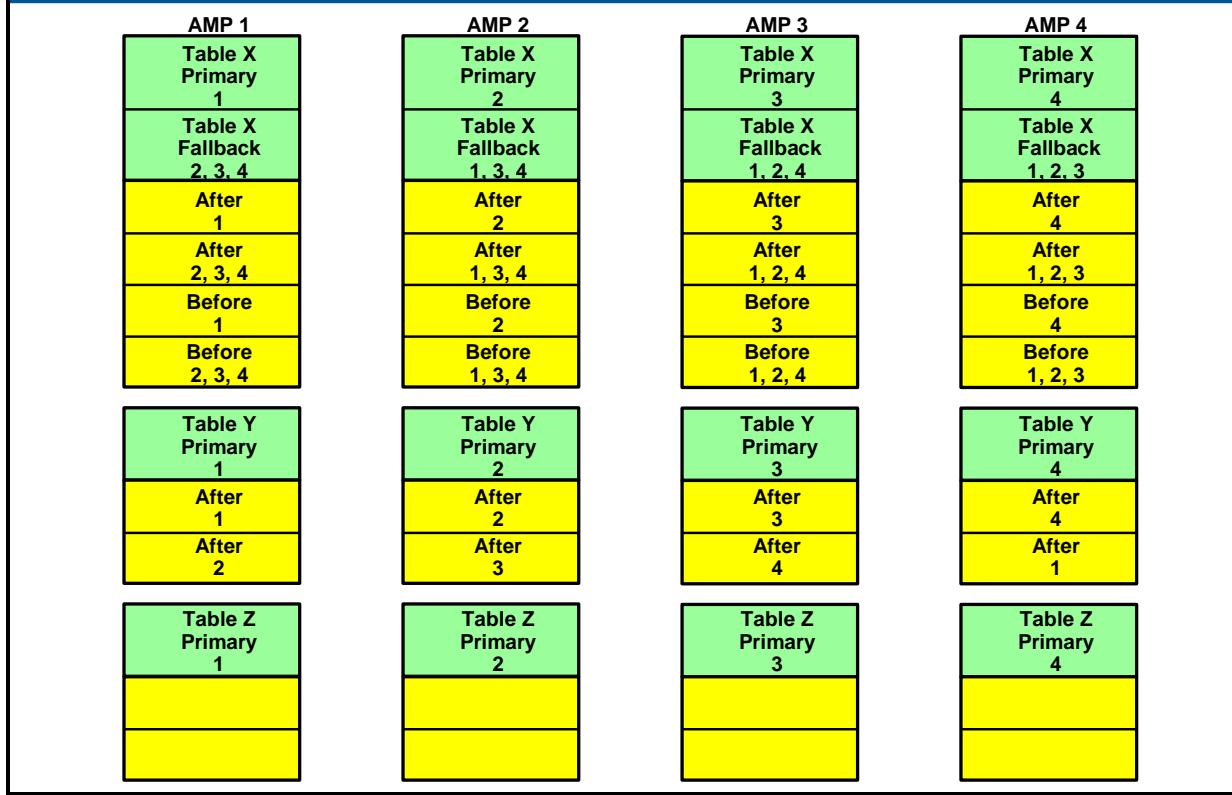
No Fallback Tables, Single Image Journals (Table Z)

- Limited processing continues
- Data is fully recovered
- Journals are lost

No Fallback Tables, No Journals

- Limited processing continues
- The administrator can only recover data to the point of the last archive

After Recovery



Summary

The facing page contains some useful concepts on how permanent journals operate during recovery.



Summary

Fallback Tables	Data is fully recoverable.
Dual Image Journals	Journals play no part in recovery.
No Fallback Tables	Data is partially available.
Dual Image Journals	Data and journals are fully recoverable.
No Fallback Tables	Data is partially available.
Single Image Journals	Data is recoverable, but journals are lost.
No Fallback Tables	Data is partially available.
No Journals	Data can be recovered only to the point of the last archive.

Teradata Training

Notes

Module 58



Archiving Data

After completing this module, you will be able to:

- Understand how to use the ARC facility to back up data on external media.
- State the access privileges needed to execute Archive and Recovery statements.
- Identify the kind of utility locks placed during archive and recovery procedures, and use statements to release the locks when appropriate to do so.
- Understand the syntax and the restrictions when archiving selected partitions of a PPI table.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Archive and Recovery Utility (ARC)	58-4
Common Uses for ARC	58-4
Archive and Recovery Phases	58-6
Restore versus FastLoad	58-8
ARC	58-10
Bakbone NetVault.....	58-10
Restart Log	58-12
Restarting Teradata ARC	58-12
Session Control	58-14
Multiple Sessions	58-16
ARC Statements	58-18
ARCHIVE Statement	58-20
ARCHIVE Examples	58-22
ARCHIVE Examples (cont.).....	58-24
Archiving Selected Partitions of PPI Table	58-26
Considerations.....	58-26
ARCHIVE Partition Example	58-28
ANALYZE Statement.....	58-30
ANALYZE Output.....	58-32
Archive Objects.....	58-34
Archive Objects (cont.)	58-36
Single Database Archive	58-36
Database ALL Archive	58-36
Single or Multiple Table Archives.....	58-36
EXCLUDE Option	58-36
Archive Levels	58-38
ALL AMP Database ARCHIVE.....	58-38
ALL-AMP Table ARCHIVE	58-38
Specific AMP or Cluster ARCHIVE	58-38
Dictionary ARCHIVE.....	58-38
Archive Levels (cont.).....	58-40
Cluster Archives.....	58-40
Archive Options	58-42
Indexes Option	58-44
Restrictions.....	58-44
Recommendations	58-44
Group Read Lock Option	58-46
Database DBC Archive	58-48
RESTORE Considerations	58-48
Summary	58-50
Review Questions	58-52

Archive and Recovery Utility (ARC)

The basic function of the Archive and Recovery (ARC) utility is to back up and optionally restore databases and database objects (e.g., tables, views, macros, stored procedures, etc.). The ARC utility performs four major tasks:

- Archive
- Restore
- Copy
- Recovery

The archive task dumps information from the Teradata system onto some type of portable storage media. The restore function reverses the archive process and moves the data from the storage media back to the database. The copy feature allows you to copy data from one system onto another system. The recovery feature utilizes information stored in permanent journals to rollback or rollforward row information.

Common Uses for ARC

The Teradata system provides a number of automatic data protection features. However, these features do not cover all types of data loss. The ARC utility provides additional data protection for the situations listed below:

- Loss of an AMP's Vdisk for no fallback tables
- Loss of multiple AMPs in the same cluster
- Failed batch processes
- Accidentally dropped tables, views, or macros
- Miscellaneous user errors
- Disaster recovery

You can use Teradata ARC to do the following:

- Archive a database, individual table, or selected partitions of a PPI table from a Teradata Database to a client resident file.
- Restore a database, individual table, or selected partitions of a PPI table back to a Teradata Database from a client resident archive file.
- Copy an archived database, table, or selected partitions of a PPI table to a Teradata Database on a different hardware platform than the one from which the database or table was archived.
- Place a checkpoint entry in a journal table.
- Recover a database to an arbitrary checkpoint by rolling it back or rolling it forward, using change images from a journal table.
- Delete change image rows from a journal table.



Archive and Recovery Utility (ARC)

Major tasks or functions of the ARC facility include:

- Archive** – captures user data on portable storage media.
- Restore** – restores data from portable storage media.
- Copy** – transfer archived data to another system or optionally back to same system
- Recovery** – recovers changes to data from permanent journal tables.

ARC provides additional data protection for these situations:

- Loss of an AMP's Vdisk for no fallback tables
- Loss of multiple Vdisks (AMPs) in the same cluster
- Failed batch processes
- Accidentally dropped tables, views or macros
- Miscellaneous user errors
- Disaster recovery

Common uses for ARC:

- Archive a database, individual table, or **selected partitions of a PPI table (V2R6)**.
- Restore a database, individual table, or **selected partitions of a PPI table (V2R6)**.
- Copy an archived database, table, or **selected partitions of a PPI table (V2R6)** to a Teradata Database on a different system.

Archive and Recovery Phases

Archive or recovery jobs always operate in two phases. The steps of each phase are described on the facing page.

The archive process is intensive. You may want to create a user just for archive activities so that you can use your user ID to perform other actions while archive is running.

Teradata ARC creates files when you archive databases, individual data tables, selected partitions of primary partition index (PPI) tables, or permanent journal tables from the Teradata Database. You provide Teradata ARC with such files when you restore databases, individual data tables, partitions of tables, or permanent journal tables back to the Teradata Database.

Teradata ARC also includes recovery with rollback and rollforward functions for data tables defined with a journal option. Moreover, you can checkpoint these journals with a synchronization point across all AMPs, and you can delete selected portions of the journals.



Archive, Restoration, and Recovery Phases

Phase 1 — Dictionary Phase

1. Allocate an event number (from DBC.Next).
2. Issue a BEGIN TRANSACTION statement.
3. Resolve object name.
4. Check access rights.
5. Place locks:
 - Utility locks on data dictionary rows.
 - Utility locks on data rows.

Note: READ locks on ARCHIVE; EXCLUSIVE locks on RESTORE.
6. Delete existing tables prior to RESTORE.
7. Issue an END TRANSACTION statement.

Phase 2 — Data Phase

1. Issue a BEGIN TRANSACTION statement.
2. Insert rows into RCEVENT and RCCONFIGURATION.
3. Perform the operation.
4. Update RCEVENT.
5. Release locks (if user specified).
6. Issue an END TRANSACTION statement.

Restore versus FastLoad

You could consider running a FastLoad utility job to restore the information to disk. This would mean that instead of archiving to tape, you have used BTEQ EXPORT or some other means to put the information into a host file for the FastLoad utility. FastLoad requires an empty table.

FastLoad Steps

Steps involved with FastLoad include:

- FastLoad uses a single session to send the INSERT statement to the PE and AMP vprocs.
- Multiple sessions are then used to facilitate sending rows to the AMP vprocs.
- Upon receipt, each AMP vproc hashes each record and redistributes it over the BYNET. This is done in parallel.
- The receiving AMP vproc then writes these rows directly to the target table as unsorted blocks.
- When loading completes, each AMP vproc sorts the target table, puts the rows into blocks, and writes the blocks to disk.
- Then, fallback rows are generated if required. FastLoad operates only on tables with no secondary indexes.
- You have to create any required indexes when the FastLoad is complete.

Restore Steps

Restoring to the same configuration includes:

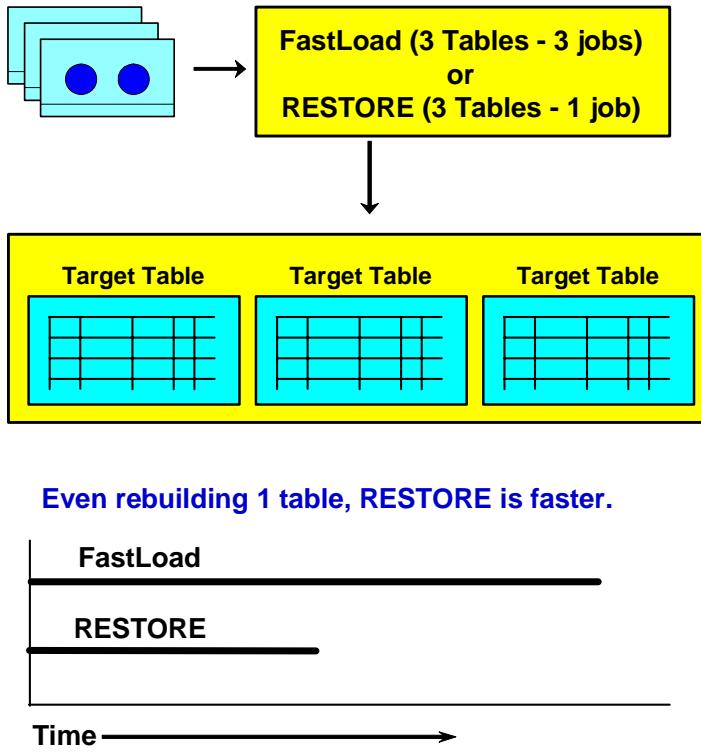
- Recovery of data blocks to the AMP vproc.
- The blocks are already in the appropriate format.

Restoring to a different configuration includes:

- The block is first sent to the AMP vproc in the old configuration.
- Then, it strips off its own rows and forwards (redistributes) the remainder of the block to the AMP vproc for the new configuration. Since the original rows were sorted in data blocks by RowID, the result is usually much faster than a normal redistribution.

ARC is the easiest and fastest to restore a very large number of objects. FastLoad operates on a table-by-table basis, while ARC can restore an entire machine with one simple command.

Restore versus FastLoad



FastLoad is a very fast loader, but not as fast as RESTORE in rebuilding a table. Why?

- FastLoad has to hash each row, redistribute every row, collect and write to disk, then read, sort, and write back to disk.
- RESTORE copies blocks to the appropriate AMPs.

Which is easier?

- FastLoad operates on a table by table basis (one at a time).
- RESTORE can restore all of the tables for one or more databases with a single job.

ARC

There are several ways to invoke the Archive facility (Release 7.0 and later).

- NetVault
- NetBackup – limited support
- Command Line (arcmain)
- Host or Mainframe

Example of a Host Job:

MVS JCL environment:

```
//JOBNAME JOB (1), 'TDAT ARCHIVE', REGION=2048K
//ARCHIVE EXEC PGM = ARCMAN, PARM='SESSIONS=10'
//STEPLIB DD DSN=DBC.SUTHLOAD, DISP=SHR
//          DD DSN=DBC.TRLOAD, DISP=SHR
//DBCLOG DD DSN=DBC.ARCCLOG, DISP=OLD
//OUTFILE DD DSN=DBC.ARCHIVE (+1), UNIT=TAPE,
//          DCB=BLKSIZE=32760,
//          DISP=(,CATLG)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD DATA, DLM=##
(Control statements go here)
##
```

Bakbone NetVault

Bakbone's NetVault is the advocated Tape Management product for direct attached Teradata BAR solutions. It is part of the Open Teradata Backup (OTB) program and has been customized for Teradata. OTB is based on off-the-shelf products from leading third party vendors along with the development of a Teradata Extension.

NetVault provides:

- The best integrated approach for Teradata
- The best archive and restore performance
- The best revenue generation offer
- The best alignment for support of BCS – Disaster Recovery
- Flexible design to support the entire enterprise



ARC

The ARC facility is required to archive/restore/copy the Teradata Database.

- ARCMAN is the program name of the Teradata ARC utility.
- ARCMAN is normally executed in batch mode, but it can be run interactively.
- Required dependency of the BAR (Backup and Recovery) backup products.
- ARC Version 8.2 is required to archive/restore/copy Teradata V2R6.2.

There are several ways to invoke the Archive facility.

- NetVault (from BakBone software)
- NetBackup (from VERITAS software) – limited support
- Command Line (execute arcmain)
- Host or Mainframe

Utilities such as NetVault allow you to create scripts, schedule jobs, and provide various tape management capabilities.

Restart Log

Since ARC does not record most of the modifications it makes to a database in the transient journal, ARC uses a file called the “restart log” to maintain the operation status of the job ARC is running. The restart log catalogs the effects of utility operations on a database or table. ARC writes restart entries in the log at the beginning and end of each significant step (and at intermediate points in a long running task). If the job is interrupted for some reason, ARC uses this log to restart the archive or restore operation that was being performed.

When you start ARC, it first places markers into the restart log to keep track of the current input and logon statements and then copies the command stream to the restart log. ARC uses these commands during restart to continue with the most recent activity using the same logon user originally entered.

Restarting Teradata ARC

The restart task execution logic differs depending on the operation, as follows.

During a Database DBC Operation

If a client system or Teradata ARC failure interrupts an archive or restore of database DBC, you cannot restart the operation. In the case of a DBC restore operation, you must perform the entire restore process again, including re-initializing the Teradata Database and the dictionary tables.

Restarting an Archive Operation

Restarting an archive operation causes Teradata ARC to do the following:

- Reposition the output archive file at the data block indicated in the last restart point recorded in the restart log before the failure occurred.
- Restart the operation at the beginning of the last table or secondary index subtable that was being archived before the failure. Teradata ARC performs this step only if the AMP configuration has changed, that is, if any of the AMPS were online before the failure but are offline after the failure.

Restarting a Restore Operation

Restarting a restore operation causes Teradata ARC to do the following:

- Reposition the input archive file at the data block indicated in the last restart point recorded in the restart log before the failure occurred.
- Restart the operation at the beginning of the last table or secondary index subtable that was being restored before the failure. Teradata ARC performs this step only if the AMP configuration has changed, that is, if any of the AMPS are online before the failure, but are offline after the failure.

Restarting a Recovery Operation

Restarting a recovery operation causes Teradata ARC to do the following:

- Resubmit the original recovery request to the Teradata Database.
- Reinitiate the complete recovery action.

The Teradata Database reads the entire input journal subtable and ignores recovery actions for change images already used.



Restart Log

Restart Log

Script	LOGON dbc/sysdba,dbapass; ARCHIVE DATA TABLES (Database_1), ABORT, RELEASE LOCK, FILE = db1_data; LOGOFF;		
Object List	Database List	Table List	
	Database_1	Table_A Table_B Table_C	

Note: The ARCHIVE facility processes databases and tables in alphabetical sequence.

Archive/recovery's first action is to write the script to the "restart log file". The log file contains:

- Current statement
- Object list
- Checkpoint positioning information
- Checkpoint configuration information

If the job is interrupted for some reason, ARC uses this log to restart the archive, restore, or recovery operation that was being performed.

The **RESTARTLOG (or RLOG) = filename** runtime parameter can be used to specify a restart log name in UNIX MP-RAS and Windows systems.

Session Control

To use the ARC utility, you must use the LOGON statement to logon to the Teradata system before you can execute other ARC statements. The user ID with which you log on has to have access rights for the ARC statements that you want to use.

The facing page shows the LOGON and LOGOFF statements.

Since the archive process can be intensive, you may want to create a user just for archiving to free your user ID for other processes while archive is running.

In general, the amount of system resources (that is, memory and processing power) that are required to support the archive or recovery increases with the number of sessions. The impact on a particular system depends on the specific configuration.

Teradata ARC uses two control sessions to control archive and recovery operations. The LOGON statement always connects these sessions no matter what type of operation being performed. Teradata ARC connects additional data sessions based on the number indicated in the SESSIONS parameter. These sessions are required for the parallel processing that occurs in archive and restore or copy operations.

If additional data sessions are required, Teradata ARC connects them at one time. Teradata ARC calculates the number of parallel sessions it can use, with maximum available being the number of sessions indicated with this parameter. Any connected sessions that are not actually used in the operation result in wasted system resources.

To request a specific number of sessions, the “SESSIONS=xnn” runtime parameter can be used. If not specified, the number of sessions defaults to 4 plus 2 control sessions for a total of 6 sessions.

The SESSIONS parameter specifies the number of Teradata Database sessions that are available for archive and recovery operations. This number does not include any additional sessions that might be required to control archive and recovery operations.



Session Control

The LOGON statement:

1. Causes two sessions to be logged on: one for SQL statements, and one for control requests.

When it encounters an ARCHIVE or RESTORE command, ARC starts additional data sessions requested in the **SESSIONS=nnn** runtime parameter.

2. Identifies the user and account to charge for used resources.
3. Identifies the user to the Teradata database system so that the software may verify ownership or check access rights. The system verifies access rights as it executes each statement.

CHECKPOINT Permits you to execute both the SQL and ARC utility checkpoint statements.

DUMP Permits you to execute the ARC Archive statement

RESTORE Permits you to execute the following ARC statements:

Restore Delete Journal Rollforward Release Lock* Rollback Build

The LOGOFF statement:

1. Ends all Teradata sessions logged on by the task, and
2. Terminates the utility.

* To release a lock held by another User, you must specify Override and hold DROP privileges on the underlying objects.

Multiple Sessions

You can specify the number of archive and/or recover sessions with which to work, or use the default. To set the number, use the SESSIONS runtime parameter.

For small systems (e.g., 5400E), the recommended number of sessions is:

- One per AMP vproc for archive.
- Two per AMP vproc for recovery.

The number of sessions to use can vary based on a number of factors. Several are described below.

The description on the facing page tells more about how the vprocs use the sessions.

If fewer than one session per vproc is specified for the archive:

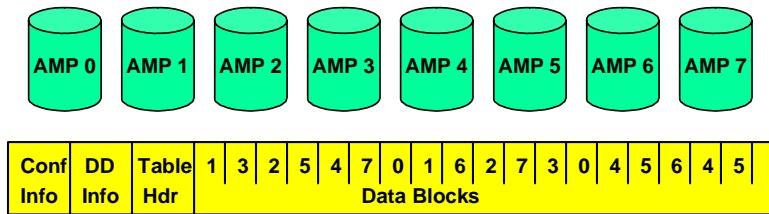
- For vproc groups, archive/recovery will archive blocks from each group with each vproc completed before the next starts.
- In this case, a large number of sessions allocated to recovery will not help recovery performance.

For larger configurations, say over 100 AMP vprocs, specifying one session per AMP will not increase performance because of other limiting component(s).

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.



Multiple Sessions



The appropriate number of sessions depends on ...

- number of AMPs
- number of channel or LAN connections
- speed and type of tape subsystem

For small systems, 1 session per AMP ensures all data blocks from all AMPs are evenly distributed.

- Teradata assigns each session to a vproc. All sessions stay with that vproc until all required data is archived. Then will it 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.

ARC Statements

The ARC utility contains a number of commands to perform archive, restore, and recovery tasks. Some of the commands are shown on the facing page.

Additional ARC options can be set via runtime parameters. The following environment variables are used with ARC.

ARCDFLT – this is the environment variable that points to the file containing the system-wide default parameters values.

Example: SET ARCDFLT=C:\TESTARC\CONFIG.ARC

The file CONFIG.ARC would include valid runtime parameters. For example:

```
SESSIONS=8  
RESTARTLOG=C:\TEMP\arcrlg1
```

ARCENV – this is the environment variable that specifies any valid Teradata ARC runtime parameters.

Example: SET ARCENV=RESTARTLOG=C:\TEMP\arcrlg2

ARCENVX – same as ARCENV, except that ARCENVX has the highest override priority. Any runtime parameter set in ARCENVX is guaranteed to be used.

Examples of typical runtime parameters that can be used include:

RESTARTLOG (or RLOG) = *filename*

The RESTARTLOG runtime option is available only on Windows and MP-RAS platforms.

Teradata ARC adds the extension type RLG to the name of the file specified in RESTARTLOG. Therefore, do not use this extension in the name of the restart log.

The restart log is created under the current directory or the working directory, if defined, unless the full path is specified.

Teradata ARC does not automatically remove the restart log files after the successful completion; therefore you may need to clean the files periodically.

SESSIONS = *nnn*

Two additional control sessions are automatically added.



ARC Statements

LOGON	Begins a session.
LOGOFF	Ends a session.
ARCHIVE	Archives a copy of a database or table to a host-resident data set/file.
ANALYZE	Reads an archive tape to display information about its content.
RESTORE	Restores a database or table from a archive file to specified AMPs.
COPY	Restores a copy of an archived file to a specified Teradata database system.
BUILD	Builds indexes and fallback data.
RELEASE LOCK	Releases host utility locks on databases or tables.
DELETE DATABASE	Deletes a database.
CHECKPOINT	Marks a journal for later archive or recovery activities.
ROLLBACK	Recovers a database and tables to a state that existed <i>before</i> some change.
ROLLFORWARD	Recovers a database or table to a state that existed <i>after</i> some change.
DELETE JOURNAL	Deletes SAVED or RESTORED Journal rows.
REVALIDATE REFERENCES	Revalidate referential integrity; a housekeeping or cleanup function.

ARCHIVE Statement

The ARCHIVE statement allows you to backup database objects to host media (usually magnetic tape). The format for this statement is shown on the following page.

Note: ARCHIVE is the preferred term as the DUMP command is supported only for backward compatibility.

The ARCHIVE control statement allows you to specify the archive:

- Type
- Objects
- Levels
- Options

The EXCLUDE option allows you to specify an alphabetical listing of database/user names that you want excluded. The values do NOT have to match existing database/user names. For example, you could EXCLUDE (A) TO (D).

Referential Integrity

Tables with *unresolved* referential integrity constraints cannot be archived. An unresolved constraint occurs when a CREATE TABLE (child) statement references a table (parent) that does not exist. Create the parent table (use the SQL command CREATE TABLE).

Effectively, create the referenced table to resolve these constraints.

ARC and HASH/JOIN Indexes

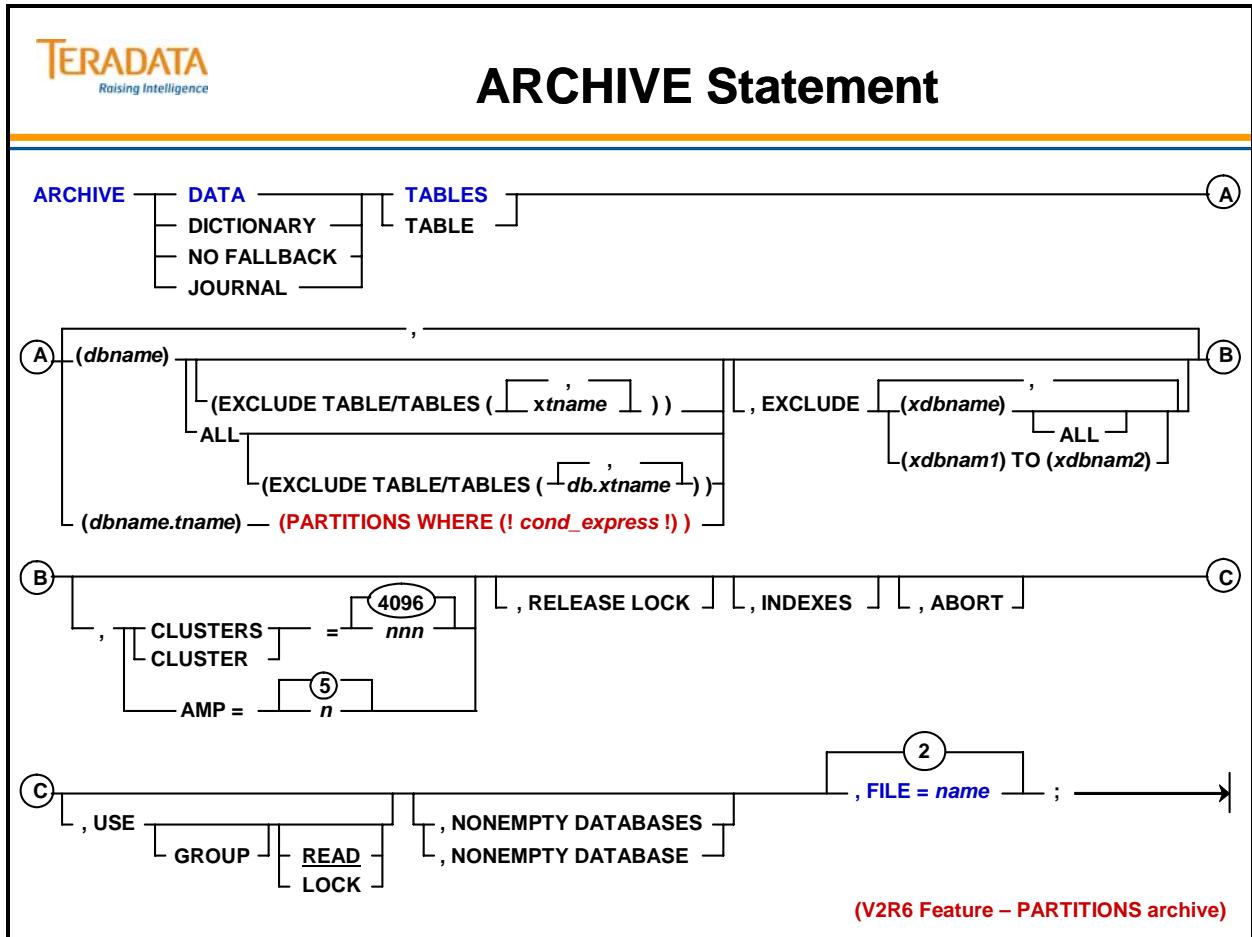
ARC cannot be used on a Hash or Join Index itself. You are permitted to archive a base table or database that has an associated Hash or Join Index defined. However, during a restore of a base table or database, the Teradata Database software does not automatically rebuild the Hash or Join Index. Instead, the Hash or Join Index is marked as invalid. You must drop and recreate the Hash or Join Index before it can be used again in the execution of queries.

The output of the SHOW HASH INDEX or the SHOW JOIN INDEX statement includes a special status message if a Join Index has been marked invalid.

A future release may lift these restrictions and provide full support for Hash and Join Indexes.

PARTITIONS WHERE

This option (new with V2R6) specifies the conditional expression for selecting partitions. If the condition selects a partial partition, the entire partition is archived.



ARCHIVE Examples

Examples of the Archive scripts are shown on the facing page.

EXCLUDE TABLE OPTION Details

This option is only accepted in a database level object in a DATA TABLES of all amps or cluster operations. A database level object that has one or more excluded tables is a partial database. An archive of a partial database contains the dictionary info and the table header row of the excluded table but actual data rows are excluded, i.e., not archived.

On the restore side, if a partial database archive is restored, no data rows will be restored for the excluded tables. If the table is excluded state, ARC restores the dictionary info and the table header row, but leaves the table in restore state. This protects the table from other application's attempt to access it before the table level restore is performed. Table level restore for the excluded tables is expected to follow the partial database restore to fully restore a partial database. If the intention of the user is to really exclude the table, the user has an option to run an explicit BUILD statement for the excluded tables. The excluded tables become accessible and are empty; they can then be dropped.

If ALL keyword is specified after the object name, then only fully qualified table names in the form of databasename.tablename is accepted in the list of EXCLUDE TABLES. If ALL is not specified then a fully qualified table name cannot be entered in the list of EXCLUDE TABLES, i.e., database names cannot be prefixed.

EXCLUDE TABLES cannot be used with the following options. ARC0215 error message will be issued if any of these conditions are detected.

Table level object: (databasename.tablename)
DICTIONARY, JOURNAL, NO FALBACK
AMP= or PN=

ARC0106: “User excluded table(s) (%s) does/do not belong to database %s”

This error is issued when a table specified in EXCLUDE TABLE list is not part of the database object. The object will be aborted, the database will be skipped and the next database will be processed.

By default, Teradata ARC counts output sectors and output rows. The row count is the number of primary data rows archived when you specify the DATA or NO FALBACK option. Both counts are included in the output listing.



ARCHIVE Examples

archive1_pd.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES (PD)
, ABORT
, RELEASE LOCK
, FILE = arc1_PD;
LOGOFF;
```

archive2_pd.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES (PD)
(EXCLUDE TABLES (dept_summary, phone_summary))
, ABORT
, RELEASE LOCK
, FILE = arc2_PD;
LOGOFF;
```

[arcmain < archive2_pd.arc](#)

Portion of output from executing above script

```
ARCHIVING DATABASE "PD"
TABLE "Department" - 3,446 BYTES, 60 ROWS ARCHIVED
TABLE "Dept_Summary" - EXCLUDED BY USER
TABLE "Employee" - 65,077 BYTES, 1,000 ROWS ARCHIVED
TABLE "Emp_Phone" - 52,504 BYTES, 2,000 ROWS ARCHIVED
TABLE "Job" - 2,898 BYTES, 66 ROWS ARCHIVED
TABLE "GT_Deptsalary" - 548 BYTES, 0 ROWS ARCHIVED
VIEW "LargeTableSpaceTotal" - ARCHIVED
TABLE "Phone_Summary" - EXCLUDED BY USER
MACRO "Set Ansi Date _ OFF" - ARCHIVED
MACRO "Set Ansi Date _ ON" - ARCHIVED
"PD" - LOCK RELEASED
DUMP COMPLETED
```

ARCHIVE Examples (cont.)

Additional examples of Archive scripts are shown on the facing page.

In the second example, Demo, Guest_Users, and Sandbox are databases or users that will be excluded from the archive.

EXCLUDE TABLE Caution

When you do a full database-level restore of an archive with excluded tables, the data dictionaries and the table headers of *all* tables, including excluded tables, are replaced.

As a result, *all* of the existing rows in the excluded tables are deleted.

You can restore individual tables from a database-level archive with excluded tables. In the RESTORE statement, you must individually specify all the tables you want to restore, except the excluded tables. By omitting the excluded tables, you preserve the data dictionaries and table headers of the excluded tables. That way you can restore the database from the archive without altering the excluded tables.

However, you cannot name macros, views, or stored procedures as objects in your RESTORE statement. So if you create an archive with excluded tables and you want to preserve the excluded tables, you cannot recover the macros, views, or stored procedures from the archive.

Archiving Large Objects (LOBs) Notes

Teradata ARC also supports the archive operation for tables that contain large object columns as long as the database systems are enabled for large object support. However, large object columns cannot be restored on a system that uses a hash function that is different than the one used for the archive.

An archive of selected partitions with LOBs is supported, but the restore is not. To restore selected partitions of LOBs, perform a full-table restore.



ARCHIVE Examples (cont.)

archive3_sysdba.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES (Sysdba) ALL
, ABORT , RELEASE LOCK
, FILE = arc3_Sys;
LOGOFF;
```

This script archives Sysdba and all of its child databases/users.

archive4_sysdba.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES
(Sysdba) ALL
(EXCLUDE TABLES (PD.dept_summary, PD.phone_summary))
, EXCLUDE (Demo), (Guest_Users) ALL, (Sandbox)
, ABORT , RELEASE LOCK
, FILE = arc4_Sys;
LOGOFF;
```

This script archives Sysdba and all of its child databases/users and excludes some tables and databases.

archive5_DBC.arc

```
LOGON dbcdbc,dbcpass;
ARCHIVE DATA TABLES (DBC) ALL
, ABORT, RELEASE LOCK
, FILE = arc5_DBC;
LOGOFF;
```

This script archives DBC and all of its child databases/users.

Archiving Selected Partitions of PPI Table

Beginning with Teradata Database V2R6.0, you can perform an all-AMPs archive on one or more partitions of a table rather than performing a full-table backup. The ability to select partitions from PPI tables is limited to all-AMP archives. Dictionary, cluster, and journal archives are not supported.

Use archive partitioning to accomplish the following tasks:

- Archive only a subset of data (this can minimize the size of the archive and improve performance).
- Restore data in a table that is partially damaged.
- Copy a limited set of data to a disaster recovery machine or to a test system.

Considerations

Consider the following when archiving selected partitions in PPI tables:

- Archiving selected partitions operates on complete partitions within tables, meaning that the selection of a partial partition implies the entire partition.
- A restore operation always deletes the selected partitions of the target table before restoring the rows that are stored in the archive.
- PPI and non-PPI tables are allowed in a single command. This allows you to manage both table types in a single database with the EXCLUDE TABLES option.
- Partitioning is based on one or more columns specified in the table definition.
- Partition elimination restricts a query to operating only in the set of partitions that are required for the query.
- Incremental archives are possible by using a partition expression that is based on date fields, which indicate when a row is inserted or updated.
- An archive or restore of selected partitions only places full-table locks. Locks on individual partitions are not supported.
- It is recommended that you re-collect table statistics after a restore of selected partitions because statistics are part of the table dictionary rows, which are not restored during a partition-level restore.
- If a table has a partitioning expression that is different from the partitioning expression used in the PPI archive, a PPI restore is possible as long as no other significant DDL changes are made to the table.



Archiving Selected Partitions of PPI Table

Starting with V2R6.0, you can perform an all-AMPs archive on one or more partitions of a table rather than performing a full-table backup and restore.

- Archiving selected partitions is limited to all-AMP archives.
- Dictionary, cluster, and journal archives are not supported.

Considerations:

- Archiving selected partitions operates on complete partitions within tables.
 - Defining a partial partition means that the entire partition will be archived.
 - A restore operation always deletes the selected partitions of the target table before restoring the rows that are stored in the archive.
- An archive or restore of selected partitions only places full-table locks. Locks on individual partitions are not supported.
- Re-collect table statistics after a restore of selected partitions because statistics are part of the table dictionary rows, which are not restored during a partition-level restore.

ARCHIVE Partition Example

An example of an Archive script that archives partitions of a PPI table is shown on the facing page.

The table definition for the Sales_PPI table is:

```
CREATE SET TABLE TFACT.Sales_PPI
(store_id      INTEGER NOT NULL,
 item_id       INTEGER NOT NULL,
 sales_date    DATE FORMAT 'YYYY-MM-DD',
 total_revenue DECIMAL(9,2),
 total_sold    INTEGER,
 note          VARCHAR(256) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX (store_id, item_id)
PARTITION BY RANGE_N (sales_date BETWEEN DATE '2000-01-01' AND DATE
'2006-12-31' EACH INTERVAL '1' MONTH );
```

Additional Notes when Archiving Partitions

Bounding condition – is well-defined if the PARTITION BY expression on the source table consists of a single RANGE_N function, and if the specified range does not include NO RANGE or UNKNOWN.

Use Correct Specifications – the incorrect use of specifications may cause the following problem. An incorrect PARTITIONS WHERE specification during backup can result in an incomplete archive or difficulties during a restore operation.

Restrict Updates to Active Partitions – it is not possible for the ARC facility to determine which partitions have been modified since the last backup. If changed partitions are not re-archived, the changes are lost when restored.

For example, if, for a given table, the backup strategy is to only backup the active (latest) partition of the table, and a change is made to a non-active partition (to fix an incorrect update), the change is not archived unless you run a separate archive of the changed partitions.

The remedy for this situation is either to restrict updates to the active partitions only (by using views to control which rows/partitions are updated) or to re-archive all modified partitions.



ARCHIVE Partition Example

archive6_ppi.arc

```
LOGON dbc/sysdba,dbapass;
ARCHIVE DATA TABLES
  (TFACT.Sales_PPI) (PARTITIONS WHERE (!Sales_Date BETWEEN '2006-01-01' AND '2006-03-31'))
  ,ABORT, RELEASE LOCK, FILE = arc6_PPI;
LOGOFF;
```

[arcmain < archive6_ppi.arc](#)

Portion of output from executing above script

```
ARCHIVE DATA TABLES (TFACT.Sales_PPI) (PARTITIONS WHERE (!Sales_Date BETWEEN '2006-01-01' AND '2006-03-31'))
,ABORT, RELEASE LOCK, FILE = arc6_PPI;
UTILITY EVENT NUMBER - 36
LOGGED ON 4 SESSIONS
ARCHIVING DATABASE "TFACT"
```

Archive Bounding Condition:

```
RANGE_N("TFACT"."SALES_PPI"."sales_date" BETWEEN DATE '2000-01-01' AND DATE '2006-12-31' EACH INTERVAL
'1' MONTH ) IN (73 TO 75 )
```

[Bounding condition is well-defined]

```
TABLE "Sales_PPI" - 2,391,241 BYTES, 40,500 ROWS ARCHIVED
"TFACT"."SALES_PPI" - LOCK RELEASED
DUMP COMPLETED
```

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. This information will help you if you are trying to restore a specific database instead of the entire archive set. This statement does not require a prior logon.

The ANALYZE statement provides the following information about the database(s) you specify:

- 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.
- If an archive file contains a selected partition archive of a table (option available starting with V2R6), the bounding condition used to select the archived partitions is displayed with an indication as to whether the bounding condition is well-defined.

DISPLAY Option

If no option is listed, display is the default. It shows the time, date and level of the archive. If you use the LONG option, the display includes the names of all tables, views, macros, triggers, or stored procedures.

VALIDATE Option

This option reads each archive record in the specified database. It checks that each data block in the file can be read but does not check whether the data block read has valid rows or not, i.e., it does not check anything inside the data block record. It only checks whether or not the data block record can be read.

You can specify both the DISPLAY and VALIDATE options on a single ANALYZE statement.



ANALYZE Statement

Format: ANALYZE [* | ALL | [(Databasename) | (Dbname1) TO (Dbname2)] [, ...]
[, DISPLAY [LONG] | , VALIDATE]
, FILE = *name* ;

Notes:

- The **ANALYZE** statement instructs the ARC utility to read an archive file and display information about its content.
- The **LONG** option displays all table, view, macro, trigger, and stored procedure names.
 - If an archive file contains a **selected partition archive of a table (V2R6)**, the bounding condition used to select the archived partitions is displayed.
- The **VALIDATE** option reads each record to check that each block on the archive file is readable.
- ANALYZE doesn't require a LOGON or LOGOFF statement.

Example: analyze1_pd.arc (script name)

ANALYZE (PD), DISPLAY LONG, FILE = arc1_PD;

To execute: arcmain < analyze1_pd.arc

ANALYZE Output

An example of the output from the ANALYZE command is shown on the facing page.



ANALYZE Output

Output from ...

ANALYZE (PD),
DISPLAY LONG,
FILE = arc1_PD;

```
:  
01/13/2006 11:30:02 CHARACTER SET IN USE: ASCII  
01/13/2006 11:30:02 ANALYZE (PD),  
01/13/2006 11:30:02 DISPLAY LONG,  
:  
01/13/2006 11:30:02 ARC VERSION 13  
01/13/2006 11:30:02 ARCHIVED AT 01-13-06 10:25:02  
01/13/2006 11:30:02 ARCHIVE CHARACTER SET: ASCII  
01/13/2006 11:30:02 ARCHIVED FROM ALL AMP DOMAINS  
:  
01/13/2006 11:30:02 RELEASE:V2R.06.01.00.02; VERSION:06.01.00.03;  
01/13/2006 11:30:02 UTILITY EVENT NUMBER - 15  
01/13/2006 11:30:02  
01/13/2006 11:30:02 DATABASE "PD"  
01/13/2006 11:30:02 TABLE "Department"  
01/13/2006 11:30:02 TABLE "Dept_Summary"  
01/13/2006 11:30:02 TABLE "Employee"  
01/13/2006 11:30:02 TABLE "Emp_Phone"  
01/13/2006 11:30:02 TABLE "GT_DeptSalary"  
01/13/2006 11:30:02 TABLE "Job"  
01/13/2006 11:30:02 VIEW "LargeTableSpaceTotal"  
01/13/2006 11:30:02 TABLE "Phone_Summary"  
01/13/2006 11:30:02 MACRO "Set Ansi Date _ OFF"  
01/13/2006 11:30:02 MACRO "Set Ansi Date _ ON"  
01/13/2006 11:30:02  
01/13/2006 11:30:02 ANALYZE COMPLETED  
:
```

Archive Objects

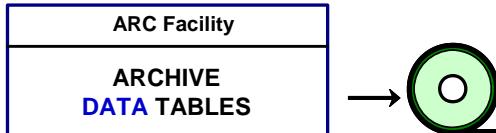
The archive statement can only back up one table type at a time: data; dictionary; no fallback; or journal. Users must submit separate archive statements in order to archive each.

Below is a description of each archive type:

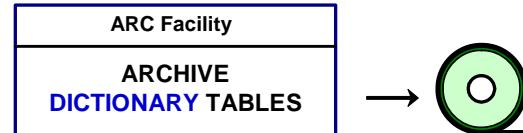
DATA TABLES	Archives fallback and non-fallback tables, views, triggers, and macros when you archive from ALL AMPs or clusters of AMPs.
DICTIONARY TABLES	Backs up DD/D rows that describe the databases or tables archived during a cluster- level archive. If you archive a database, the archive includes table, view, trigger, and macro definitions. If you archive a table, back up only includes table definition rows. DD/D information for permanent journals is not included.
NO FALBACK 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.
JOURNAL TABLES	Archives the dictionary rows and selected contents of the journal tables.



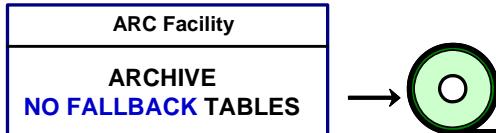
Archive Objects



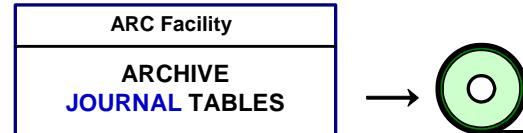
- Fallback Tables, Views, Macros, Triggers, and Stored Procedures
- No fallback tables
- All AMP or cluster archive



- DD/D rows to complement cluster-level archive



- Non-fallback tables
- Archives AMP data missed during previous all AMP or cluster-level archive



- Journal Tables

Archive Objects (cont.)

The information backed up in an archive operation varies depending upon the type of object you select:

- Single database or table
- Multiple databases or tables
- All databases

Single Database Archive

An ALL AMP database archive backs up a wide range of DD/D information. It archives all objects that belong to the database including views, macros and the data tables themselves. The information archived for the data tables includes table, column, and index information as well as table headers and data rows. A table header is a row of information about the table that is kept in the first block of the table.

Database ALL Archive

A Database ALL archive archives the parent and all children. The backed up objects are identical to those archived in a single database archive.

Single or Multiple Table Archives

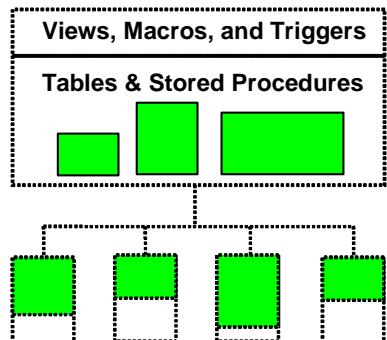
For each table specified in the archive statement, the ARC utility backs up table, column, and index information along with table headers and the actual data rows.

EXCLUDE Option

This option directly affects which databases are backed up. The exclude option changes the range of objects that the ARC utility archives. Users can leave out a single database, a database and all of its children, or a range of alphabetically sorted databases.

Archive Objects (cont.)

An ALL AMP archive that identifies a database that contains data tables, etc.



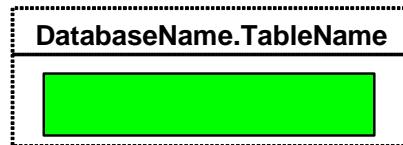
Archives all DD/D information for the identified database, including views, macros, triggers, and stored procedures. The archive also includes all table, column and index information, as well as table headers and data rows.

The ALL option archives all items listed above for the specified database, as well as all its descendants.

Note:

The EXCLUDE option allows you to exclude a single database, a database and all its descendants, a range of alphabetically sorted databases, or specific tables.

An ALL AMP archive that identifies a single table.



An ALL AMPs table archive that identifies a table archives table, column and index information, as well as table headers and data rows.

Archive Levels

There are four levels of archives that archive different information to a removable media.
(See the list on the facing page.)

ALL AMP Database ARCHIVE

An ALL AMP Database archive contains:

- Data rows from all the Tables in the Database(s) being archived.
- The Data Dictionary/Directory rows of the object(s) being archived.
- All Table, View, and Macro information.
- Information about the structure of all the Tables in the Database(s).

You can restore a table from an archive only if the table ID in the archive matches that in the DBC.TVM table. To restore a database or user, the database ID in the archive must match the database ID in the DBC.Dbase table. You can only restore database DBC to an otherwise empty Teradata database.

ALL-AMP Table ARCHIVE

An all-AMP table archive contains:

- Data rows from the table
- Dictionary information for the table
- All table, column, and index definitions
- Table structure information

Specific AMP or Cluster ARCHIVE

Specific AMP or cluster archives include:

- Data rows from the table or tables within the database(s)
- No data dictionary rows residing on that AMP or within that cluster
- Table structure information
- A supplemental data dictionary archive to provide necessary information (for dictionary archives only)

Dictionary ARCHIVE

A dictionary archive contains:

- Dictionary rows from the database DBC for the archived object(s)
- No permanent journal information



Archive Levels

ALL AMP Database ARCHIVE includes:

- Data rows from the tables in the specific database(s).
- Table structure information.
- All table, column, and index definitions.
- All views, macros, and triggers definitions.
- Stored procedures.
- Permanent journal information is not included.

ALL AMP Table ARCHIVE includes:

- Data rows from the table.
- All dictionary information for the table.
- All table, column, and index definitions.

Specific AMPs or Cluster ARCHIVE includes:

- Data rows from the table or tables within the specific database(s).
- No dictionary rows.

Dictionary ARCHIVE includes:

- Dictionary rows for the object being archived.
(Tables: TVM, TVFields, Indexes, IndexNames.)
- Permanent journal information is not included.

Note:

Since a Cluster ARCHIVE does not contain dictionary information, you must maintain a Dictionary archive to restore the database or tables.

Archive Levels (cont.)

The default archive level for any archive operation is all vprocs.

Normally, you do not specify an archive level in your ARCHIVE statement since ALL is the default. When an AMP vproc is off-line during an all-AMP archive, non-fallback tables may only be partially archived.

You need to perform a single-AMP back up of NO FALBACK TABLES to obtain a complete back up. Fallback tables are always completely archived even if a vproc is down, because there is either a primary or fallback copy of the data on another AMP vproc.

Cluster Archives

As an alternative to archiving data tables from all AMPs into a single archive, you can partition the archive into a set of archive files called a cluster archive. A cluster archive archives data tables by groups of AMP clusters so that the complete set of archive files contains all data from all 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 vproc 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 vproc.

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 Database DBC is automatically excluded for 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.

Cluster archives have two restrictions:

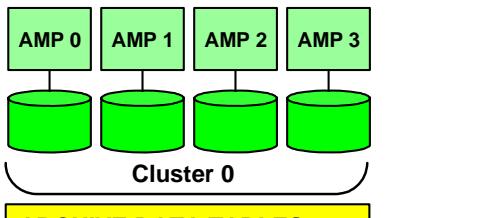
- You cannot create a cluster archive of journal tables.
- You cannot setup cluster archives when you are archiving DBC database.

Archive Levels (cont.)

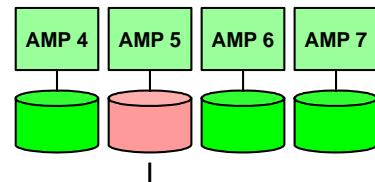
The system performs an ALL AMP level archive unless you specify a processor or cluster archive.

You can partially archive non-fallback tables if an AMP is offline.

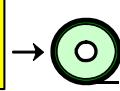
Fallback tables are always completely archived, regardless of the configuration.



**ARCHIVE DATA TABLES
(DBC) ALL ,
CLUSTER = 0,
FILE = cluster0;**



**ARCHIVE NO FALLBACK
TABLES (DBC) ALL ,
AMP = 5,
FILE = amp5only;**



Cluster level archives group data from one or more clusters into separate archive data sets.

Cluster archive jobs can be run in parallel or scheduled to be run at different times.

A single AMP can be recovered in less time.

Dictionary information must be archived separately.

Single AMP archives are only used to complete the archive of no fallback tables after the AMP is returned to service.

Archive Options

The archive statement includes a number of options. Each option is described below:

RELEASE LOCK	Automatically releases Utility Locks if the operation completes successfully.
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.
ABORT	Causes all AMP or cluster archives to fail with error messages if an AMP is off-line and the objects to be archived includes: <ul style="list-style-type: none">• No fallback tables• Single image journals
NONEMPTY DATABASES	Instructs the ARC utility to exclude users/databases without tables, views, macros, or triggers from the archive.
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.



Archive Options

- **Release Lock**
 - Utility locks automatically released upon successful operation completion
- **Indexes**
 - Restricted to all-AMP dumps
 - Includes secondary indexes with archive
 - Requires more time and media
- **Abort**
 - Fails ALL AMP or cluster dumps AND provides error messages if:
 - > AMP vproc is off-line AND,
 - > Dumped objects include no fallback tables, OR
 - > Dumped objects include single-image journals
- **Non empty Database(s)**
 - Excludes users/databases without tables, views, macros, triggers, or stored procedures from archive operation
- **Use Group Read Lock**
 - Permits concurrent table archiving and transaction updates on locked rows
 - Requires after-image journaling of table

Indexes Option

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.

The INDEXES option archives both unique and non-unique secondary indexes on all data tables. However, if an AMP vproc is off-line, the utility only archives unique secondary indexes on fallback tables. It ignores the non-unique 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 on-line.

The reverse process is true for restoring data that was archived with the INDEXES option. All indexes are restored if all AMPs are on-line. If an AMP is down, only unique secondary indexes are restored and only for fallback tables. No non-unique secondary indexes are restored. No indexes are restored for non-fallback tables.

Restrictions

You can only use the INDEXES option with all-AMP data table archive operations. The INDEXES option does not apply to dictionary, no fallback, and journal table archive operations. This option is ignored in cluster or single processor archive operations as well as an archive statement that includes the GROUP READ LOCK option.

Recommendations

If you specify the INDEXES option, the time and media required to perform and archive increases. It will also take you longer to restore an archive you have created with the INDEXES option than to restore an archive created without it. However, it will usually be quicker to restore secondary indexes than rebuild them. In most cases, archive and restore without INDEXES.

The following do not archive index subtables:

- Dictionary, no fallback, or journal table archives
- Cluster or single processor archives
- Archives made using a group read lock



Indexes Option

This option applies only to ARCHIVE DATA TABLES ALL AMPs.

ARCHIVE Operation

If all AMPs are online, then all indexes are archived.

otherwise

If the table is fallback, then only unique secondary indexes are archived.

otherwise

No indexes are dumped.

RESTORE Operation

If all AMPs are online, then all indexes are restored.

otherwise

If the table is fallback, then only unique secondary indexes are restored.

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. Only tables that have an after-image journal associated with them can utilize the group read lock option. This option is only valid during an all-AMPs archive.

The ARC utility places a read lock on tables during archive operations that prevent 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 the READ LOCK option to circumvent this limitation using the following steps:

- The utility places an access lock on the entire table.
- A group of table rows are read locked (about 64 KB).
- The locked rows are archived.
- The lock is released on that group of rows.
- Another group of rows is locked... etc.

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 which disables writings on a small group of rows within the table, users can continue to make updates directly to the rows not being archived. In the event that someone attempts to update a row that is under a read lock, the change is written to the after-image journal but the data row remains unchanged in the archive copy. The after-image journal must be backed up to have a complete archive.

Example

The facing page illustrates an archive process with the group read lock option. The shaded rectangle indicates the rows containing a read lock. Any changes submitted to rows 011 through 100 will not be written directly to the archive of the table. Three transactions occurred during the archive process. The first transaction affected row 001. This change is not reflected in the archive file since it occurred after that row was already archived. The second transaction affected row 080. This change is not in the archive file either because it had a read lock on it when the transaction occurred. The third transaction affected row 101. This transaction will appear in the archive file since it took place before row 101 was archived.

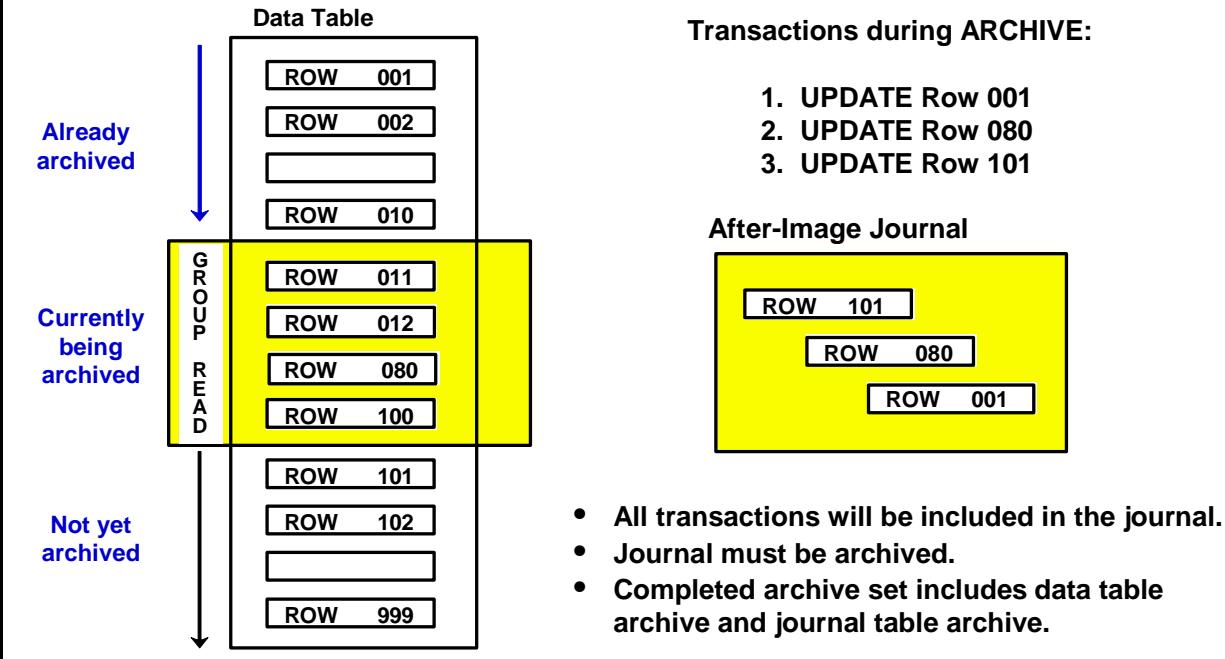
All three transactions are written to the after-image journal table. Once the archive is complete, the user will archive the after-image journal. The data table archive along with the journal table archive represents a complete archive of the data.

Requirements and Restrictions

- The backup must be an all-AMP or cluster-level archive.
- You cannot archive system user DBC User with GROUP READ LOCK.
- The table must have after-image journal and the journal must be archived to complete the backup.

Group Read Lock Option

- Rolling archive while you are using a table.
- Must have After-Image Journal defined.
- Only valid for an all-AMPs archive.



Database DBC Archive

An archive of the information in DBC should be done every time DDL makes changes to the definitions stored in the database. Examples of the types of commands that make these changes are:

- CREATE DATABASE/USER
- MODIFY DATABASE/USER
- CREATE/ALTER TABLE
- CREATE/REPLACE VIEW
- CREATE/REPLACE MACRO
- CREATE INDEX
- DROP TABLE/VIEW/MACRO
- DROP INDEX
- GRANT
- REVOKE

Database SYSUDTLIB is linked with database DBC and is only archived if DBC is archived. SYSUDTLIB cannot be specified as an individual object in an **ARCHIVE** statement.

If database DBC is involved in an archive operation, it is always archived first, followed by database SYSUDTLIB. If additional databases are being archived, they will follow SYSUDTLIB in alphabetical order.

RESTORE Considerations

If you drop a table in a database, you cannot restore a dropped table unless you restore the entire database.

Furthermore, you cannot restore a dropped database unless you restore database DBC first.

If you need to restore all of a user database or database DBC (that is, all of the Teradata Database) because of a catastrophic event, you can restore the dictionary information for the database at the database level before you restore the individual tables. Restoring the dictionary first restores the table definitions, so you are able to successfully restore the tables.

You can only restore Database DBC to an initialized Teradata Database – usually following a SYSINIT. An initialized Teradata system can only have the user DBC and the default users of ALL, Default, and Public in order to RESTORE DBC (ALL).



Database DBC Archive

An archive of database DBC causes the system to copy the following tables to the archive.

AccessRights	Specification of all GRANTed rights
AccLogRuleTbl	Stores access logging specifications
Accounts	Lists all authorized account numbers
CollationTbl	Defines MULTINATIONAL collation
DBase	Definition of each DATABASE and USER
Hosts	Character set default override rules
LogonRuleTbl	User, host, password requirements
Next	Internal table for generating TABLE and DATABASE identifiers
OldPasswords	Lists passwords that are no longer in use,
Owners	Defines all databases owned by another
Parents	Defines the parent/child relationship between databases
Profiles	Defines Profiles
RCConfiguration	Records the configuration for the RCEvents rows
RCEvent	Records all archive and recovery events
RCMedia	Records all removable media used in archive activities
RepGroup	Defines each replication group in the server.
Roles	Defines Roles
RoleGrants	Contains Users and Roles granted to Roles
Translation	National character support tables
UDTCast	Contains source & target data types used in casting operations for UDTs.
UDTInfo	Captures the specifics contained within the CREATE TYPE statement.
UDTTransform	Contains the transform group name and the routine identifiers.

Database DBC can only be restored to an initialized Teradata Database system.

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- **Archive and Recovery (ARC)** is a command-line utility that performs three operations: archive, restore and recovery.
- For small systems, the optimum number of sessions for archive and recovery operations is:
 - One per AMP vproc for archive
 - Two per AMP vproc for recovery
- An archive operation can back up a single database or table, multiple databases or tables, or all databases.
- Available archive levels are all-AMP, specific AMP and cluster archives.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

- 1. True or False.** The Archive and Recovery utility protects against more types of potential data loss than automatic data protection features.
- 2. True or False.** Recovery and FastLoad are about the same in ease and speed to recover data.
- 3. True or False.** An All-AMPs archive of a database archives all of the objects in the database.
- 4. True or False.** Archiving a partition of a PPI table places a partition-level lock on the partition being archived.

Teradata Training

Notes

Module 59



Restoring Data

After completing this module, you will be able to:

- Use the ARC facility to replace existing data on a Teradata system with information stored on portable storage media.
- Understand the RESTORE, COPY, BUILD, REVALIDATE REFERENCES FOR, and RELEASE LOCK statements.
- Use Recovery Control views to obtain ARC event information.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Understanding Restore Operations	59-4
Considerations before Restoring Data	59-4
Restore-Related Statements	59-6
The Restore Statement	59-8
Restoring Examples	59-10
RESTORE Example and Output.....	59-12
EXCLUDE TABLE Caution:	59-12
Restoring Selected Partitions of PPI Table	59-14
RESTORE Partition Example	59-16
COPY Statement	59-18
Copying Partitioned Data.....	59-18
Copying Objects.....	59-20
Keyword Options with COPY	59-20
Copying	59-22
BUILD Statement	59-24
RELEASE LOCK Statement	59-26
Revalidate References.....	59-28
Revalidate References Output.....	59-30
Recovery Control Data Dictionary Views	59-32
DBC.Association[X] Views.....	59-32
DBC.Events[X] Views.....	59-32
DBC.Events_Configuration[X] Views	59-32
DBC.Events_Media[X] Views.....	59-32
Association View	59-34
Events View	59-36
Restoring Data Summary	59-38
Review Questions	59-40

Understanding Restore Operations

A restore operation transfers database information from archive files backed up on portable storage media to all AMP vprocs, clusters of AMPs, or specified AMP vprocs.

Considerations before Restoring Data

Before performing a restore operation, consider the following items.

Dropped Database and Users – a restore of a database DBC drops all new databases or users created since the time the archive was created.

Dropped Tables, Views, and Macros – a restore of a user database drops any new tables, views, macros, stored procedures, and triggers created since the archive of the database.

Restoring Undefined Tables with COPY – because of potentially conflicting database and table internal identifiers, you cannot restore a database or table to another system that does not contain an equivalent definition of the entity (for example, the same name and internal identifier). To restore data tables that are not already defined in the data dictionary, use the COPY statement.

Insufficient Memory for Large Tables – Teradata ARC uses the same methodology as the Teradata SQL CREATE INDEX function to rebuild secondary table indexes. If there is insufficient available disk space, it may not be possible to restore a very large table because of the amount of temporary disk space that is required to recreate a secondary index.

Join Indexes – Teradata ARC does not archive or restore join indexes. If a database containing a join index is restored, then the join index will no longer exist when the restore operation is complete. If a partial database restore is done where a table is restored, any join indexes that reference that table will be marked as invalid.

A warning occurs when a SHOW JOIN INDEX request is performed on an invalidated join index. A HELP DATABASE request lists both valid and invalid join indexes, but it will not indicate which join indexes are invalid. To re-create the join index, do the following:

1. Extract the join index definition from the SHOW JOIN INDEX output.
2. Drop and re-create the join index.
3. Collect new statistics.

Matching Hash Functions for Large Objects – Teradata ARC supports the restore operation for tables that contain large object columns as long as the database system is enabled for large object support. However, large object columns cannot be restored on a system that uses a hash function that is different than the one used for the archive.

Certain Statements Force the Restoration of the Entire Database – a Teradata SQL DROP or RENAME statement cause the definition of an entity to be removed from the dictionary, and this same definition cannot be re-created using a Teradata SQL CREATE statement because a create operation is a new logical definition.



Understanding Restore Operations

Restore operations transfer information from archive files to AMP vprocs.

Data Definitions

- Database archives contain dictionary definitions.
- Dictionary table archives contain dictionary definitions.

Replacing Objects

- ALL AMP vproc archives contain data and dictionary definitions.
- Restore operations replace both.

Notes:

- You can only RESTORE an entity if the Data Dictionary has an equivalent definition of the entity being restored (same name and internal ID).
- The COPY operation can be used if the object doesn't exist. To COPY the object in a database/user, the database/user must exist on the target system.

Restore-Related Statements

The Archive and Recovery utility provides several recovery control statements you use during restore-related operations. Each command is described below:

ANALYZE	Reads the contents of an archive tape and displays the information.
BUILD	Builds 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	Copies a database or table from an archived file to the same or different Teradata system than the one from which it was archived.
DELETE DATABASE	Deletes all data tables, views and macros from the database. This command does not remove journal tables.
RELEASE LOCK	Removes a utility lock from a specific database or table.
RESTORE	Moves data from archive files back to the same Teradata system from which it was archived. You can also restore data to another system. For example, migrating from V1 to V2 required a fresh system (Sysinit).
REVALIDATE REFERENCES FOR	Validates inconsistent restraints against a target table thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables.

You may invoke the Archive and Recovery utility from a channel-attached MVS or VM host system.



Restore-Related Statements

LOGON	Begins a session.
LOGOFF	Ends a session and terminates the utility.
ANALYZE	Reads an archive tape to display information about its contents.
RESTORE	Restores a database or table from an archive file to specified AMP Vprocs.
COPY	Restores a copy of an archived file to a specified Teradata database System.
BUILD	Builds Indexes and fallback data.
RELEASE LOCK	Releases host utility locks on databases or tables.
DELETE DATABASE	Deletes data tables, views and macros from a database.
REVALIDATE REFERENCES	Validates inconsistent restraints against a target table thereby allowing users to execute UPDATE, INSERT and DELETE statements on the tables.

The Restore Statement

The RESTORE statement allows you to replace database objects from an archive tape to the same system or to another system. The ARC facility has four types of restore or recover operations described below:

Data Tables

The DATA option restores fallback, non-fallback, or both types of data tables to all AMP vprocs or clusters of AMP vprocs. If you restore a database, the data dictionary definitions (for table, view, macro, and triggers) are restored automatically. If you restore a table, only table definition rows are included.

Dictionary Tables

The DICTIONARY option restores data dictionary rows that describe the databases or tables dumped (necessary with a cluster-level restore). A RESTORE DICTIONARY TABLES only restores the definitions of all the entities in the dictionary for the selected databases.

No Fallback Tables

Use the no fallback option to restore a single processor.

Journal Tables

This option restores an archived journal for subsequent use in a roll operation.

Restore Fallback

This option applies only to data table restores of fallback tables. This option restores the fallback copy of primary and unique secondary indexes while restoring the data.

No Build

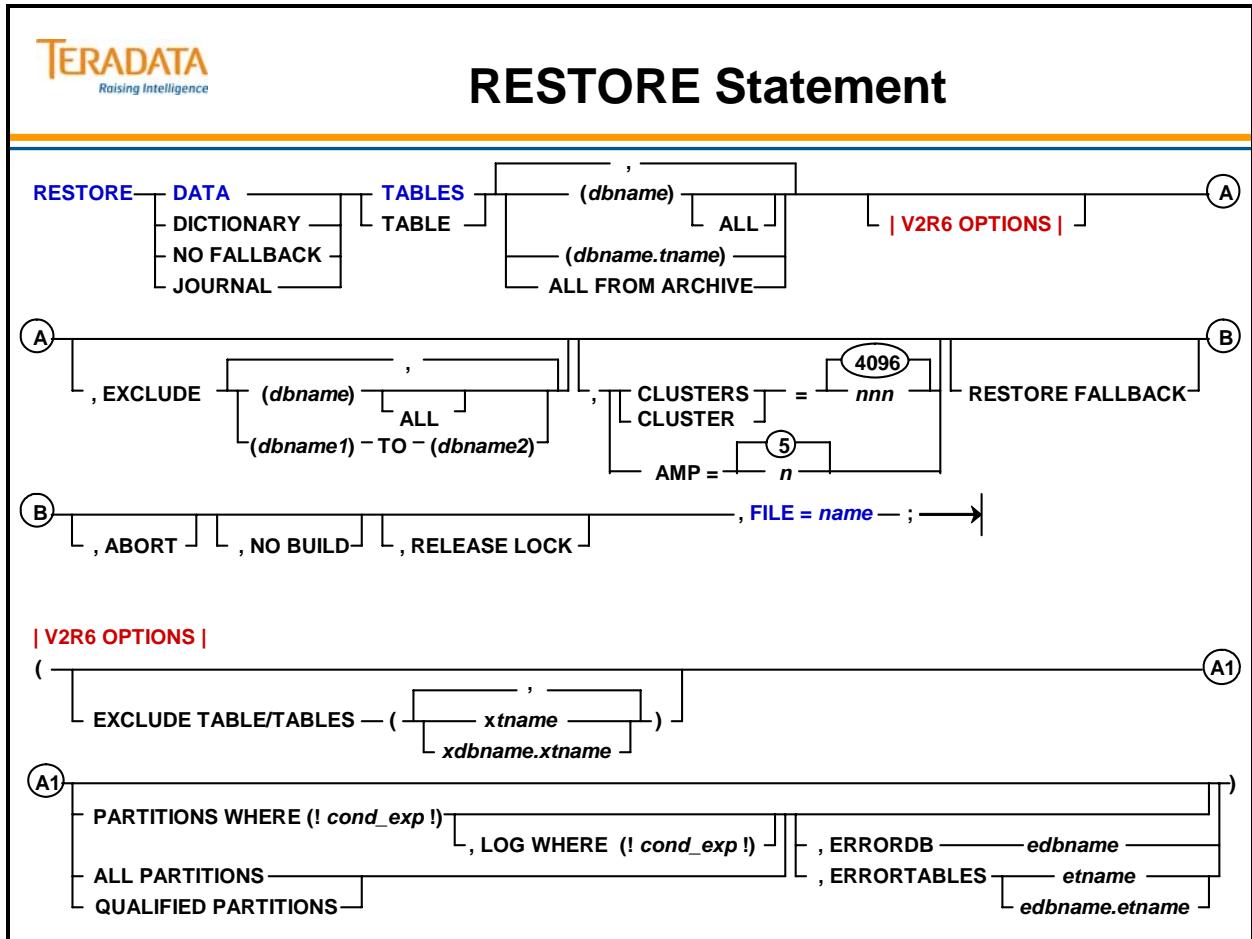
NO BUILD prevents secondary indexes on non-fallback tables from being restored or built. On fallback tables, it prevents the creation of secondary indexes and fallback table rows.

Release Lock

Automatically release of the utility locks when a restore completes successfully.

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.



Restoring Examples

Three RESTORE examples are shown on the facing page.

As mentioned before, you can restore archived data tables to the Teradata Database if the data dictionary contains a definition of the entity (same name and same internal ID) you want to restore.

For example, if the entity is a database, that database must be defined in the dictionary. Or, if the entity is a table, that table must be defined in the dictionary. You cannot restore entities not defined in the data dictionary.

A dictionary table archive contains all table, view, and macro definitions in the database. A restore of a dictionary archive restores the definitions of all data tables, views and macros. However, it does not restore any data.

ALL FROM ARCHIVE

In the first example, the ALL FROM ARCHIVE keywords take the place of the database and/or table names that are normally specified after the DATA, DICTIONARY, JOURNAL, or NO FALBACK TABLES keywords.

You are not allowed to specify any other database or table names to be restored when using ALL FROM ARCHIVE. All databases and tables in the given archive file will be restored, and any existing databases or tables will be overwritten.

ALL FROM ARCHIVE can not be used to restore database DBC. The user must exclude DBC if it is present in the archive being restored.

Specified Database

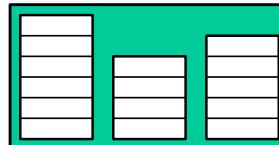
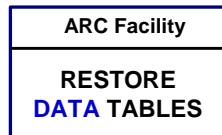
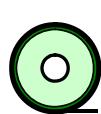
The second example is restoring the SYSDBA database when all AMP vprocs are online. The restore type is data and the restore object is all databases belonging to user SYSDBA. Since there is no mention of any restore levels, such as a specific AMP number, the system assumes all AMPS. The release lock option removes the utility lock after completing the restore operation. The name of the archive file is “arch3_Sys”.

The third example has a narrower scope. This statement is only restoring non-fallback tables on AMP 1. The administrator has already performed an all-AMPS restore on the rest of the system. The release lock option removes the utility lock after completion of the restore operation. The archive filename is “arch3_Sys”.

Any databases or users created since the Archive of the dictionary or any table, view, or macro created since the archive of a database will be dropped when you restore the DBC database or a user database.



RESTORE Examples

DATA
TABLE(S)**restore1_pd.arc**

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES ALL FROM ARCHIVE
, RELEASE LOCK, ABORT, FILE = arc1_PD;
LOGOFF;
```

Restore all tables and databases from archive (ALL AMPs restore with all AMPs online).

restore3_sysdba.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES (SYSDBA) ALL
, RELEASE LOCK, ABORT, FILE = arc3_Sys;
LOGOFF;
```

Restores all objects for Sysdba and ALL child databases/users of Sysdba (ALL AMPs restore with all AMPs online).

restore_amp1.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE NO FALBACK TABLES (SYSDBA) ALL
, AMP=1, RELEASE LOCK, FILE = arc3_Sys;
LOGOFF;
```

Restores non-Fallback tables for Sysdba and ALL child databases/users of Sysdba (single AMP restore - AMP 1).

RESTORE Example and Output

An example of the output from a RESTORE command is shown on the facing page.

EXCLUDE TABLE Caution:

When you do a full database-level restore of an archive with excluded tables, the data dictionaries and the table headers of *all* tables, including excluded tables, are replaced.

As a result, *all* of the existing rows in the excluded tables are deleted.

You can restore individual tables from a database-level archive with excluded tables. In the RESTORE statement, you must individually specify all the tables you want to restore, except the excluded tables. By omitting the excluded tables, you preserve the data dictionaries and table headers of the excluded tables. That way you can restore the database from the archive without altering the excluded tables.

However, you cannot name macros, views, or stored procedures as objects in your RESTORE statement. So if you create an archive with excluded tables and you want to preserve the excluded tables, you cannot recover the macros, views, or stored procedures from the archive.



RESTORE Example and Output

restore1_pd.arc

```
LOGON dbc/sysdba,dbapass;  
RESTORE DATA TABLES  
    ALL FROM ARCHIVE  
    , ABORT  
    , RELEASE LOCK  
    , FILE = arc1_PD;  
LOGOFF;
```

Output from this restore script ...

```
... : RESTORE DATA TABLES ALL FROM ARCHIVE
... , ABORT
... , RELEASE LOCK
... , FILE = arc1_PD;
... UTILITY EVENT NUMBER - 17
... LOGGED ON 4 SESSIONS
... STARTING TO RESTORE DATABASE "PD"
... "LargeTableSpaceTotal" - VIEW RESTORED
... "SetAnsiDate_OFF" - MACRO RESTORED
... "SetAnsiDate_ON" - MACRO RESTORED
... DICTIONARY RESTORE COMPLETED
... "Department" - 3,446 BYTES, 60 ROWS RESTORED
... "Dept_Summary" - 2,966 BYTES, 50 ROWS RESTORED
... "Employee" - 65,077 BYTES, 1,000 ROWS RESTORED
... "Emp_Phone" - 52,504 BYTES, 2,000 ROWS RESTORED
... "GT_Deptsalary" - 548 BYTES, 0 ROWS RESTORED
... "Job" - 2,898 BYTES, 66 ROWS RESTORED
... "Phone_Summary" - 51,464 BYTES, 1,960 ROWS RESTORED
... "PD" - LOCK RELEASED
...
... STATEMENT COMPLETED
```

Restoring Selected Partitions of PPI Table

Selected partitions can be directly backed up and restored with a PARTITIONS WHERE option that restricts the list of rows processed. The PARTITIONS WHERE option operates on complete table partitions. A RESTORE (or COPY) completely wipes out selected partitions (specified by the PARTITIONS WHERE option) of an existing target table before recovering the rows stored on the backup tape.

A restore of selected partitions is impacted by the various maintenance activities that can occur on a table. For example, a user may not be able to perform a full-table backup every time a secondary index is added or dropped, or the partitioning expression is changed. A restore of selected partitions is able to restore data into a target table with different characteristics than the source stored on tape.

To RESTORE selected partitions, a table must already exist on the target system.

PARTITIONS WHERE Keyword

Use the PARTITIONS WHERE option to specify the conditional expression, which contains the definition of the partitions that you want to restore. The following restrictions apply to the use of PARTITIONS WHERE option:

- The object must be an individual table name (not a database).
- The source and target tables must have a PARTITIONS BY expression defined.
- The restore is an all-AMP restore (not a dictionary, cluster, or journal restore).
- If the table belongs to a database that is specified in the RESTORE statement, the table is excluded from the database-level object (with EXCLUDE TABLES) and is individually specified.
- Any name specified in the conditional expression is within the table specified.
- It is recommended that the only referenced columns in the conditional expression be the partitioning columns or system-derived column PARTITION of the table. References to other columns do not contribute to partition elimination, and might accidentally qualify more partitions than intended.

LOG WHERE Keyword

You might find that the PARTITIONS WHERE option does not capture all the rows that need to be restored. In this case, use the LOG WHERE option to insert into a Teradata-generated error table archived rows that both fall outside the partitions specified by the PARTITIONS WHERE conditional expression and match the LOG WHERE conditional expression.

Use the option only if PARTITIONS WHERE is also specified for the object. If LOG WHERE is omitted, the default is to log to the error table only the rows in the partitions being restored that have errors.



Restoring Selected Partitions of PPI Table

PARTITION Options with RESTORE and COPY commands.

PARTITIONS WHERE (! conditional expression !)

This option specifies a conditional expression that contains the definition of the partitions that you want to restore/copy.

LOG WHERE (! conditional expression !) – the conditional expression specifies rows to log to the error table when restoring selected partitions.

ERRORDB / ERRORTABLES – specifies the location of the error log for partition-level operations.

ALL PARTITIONS

Use this option to restore/copy all of the archived partitions for a PPI table.

QUALIFIED PARTITIONS (may not be used very often)

Use this option only to restore/copy a specific-AMP archive after restoring selected partitions from an all-AMP archive done while an AMP is down.

Note: For partition archives, specify PARTITIONS WHERE or ALL PARTITIONS.

If PARTITIONS WHERE or ALL PARTITIONS options **are not specified** for a RESTORE or COPY operation, **the default action is to overwrite the entire table with the archived table definition and data.** Essentially, this is the same as a full-table restore.

RESTORE Partition Example

An example of a Restore script that restores all partitions in a partition archive for a PPI table is shown on the facing page.

The partitioning expression for the Sales_PPI table is:

```
PARTITION BY RANGE_N (sales_date BETWEEN DATE '2000-01-01' AND DATE  
'2006-12-31' EACH INTERVAL '1' MONTH );
```

Additional Notes when Restoring Partitions

Always Specify PARTITIONS WHERE or ALL PARTITIONS – If the PARTITIONS WHERE or ALL PARTITIONS options are not specified for a RESTORE or COPY operation, the default action is to overwrite the entire table with the archived table definition and data. Essentially, this is the same as a full-table restore.

For example, if you forget to use PARTITIONS WHERE when you try to restore a single partition backup, data is dropped from the table and the single partition stored on the archive is restored.

Know What Partitions are Being Deleted – with a RESTORE or COPY operation, all partitions that match the PARTITIONS WHERE condition are deleted, even if they are not stored in the archive.

For example, if you restore an archive that contains the data for April 2006, but mistakenly enter a PARTITIONS WHERE condition that matches both March and April 2006, the data for both March and April 2004 are deleted, and only April 2006 is restored.

The remedy for this situation is to be very careful about using the PARTITONS WHERE condition. If there is any doubt about which partitions are affected, COPY the selected partition backup to a staging table, and manually copy the desired partition(s) into the target table using INSERT/SELECT and/or DELETE.

Avoid Restoring From a Previous Partitioning Scheme – when changing the partitioning expression for a table, it is possible to change the boundaries of existing partitions. If these partitions are restored, Teradata might either drop more data than expected or restore less data than expected, if the archive does not include data for all of the selected partitions.

For example, if an archive is done on a table partitioned by month with the archive data corresponding to March 2006, and the table is re-partitioned by week, then a PPI restore of the March backup (using ALL PARTITIONS) overwrites the data for all weeks that contain at least one day in March. As a result, the last few days of February and the first few days of April might be deleted and not restored.

The remedy for this situation is to avoid restoring partition backups from a previous partitioning scheme to an updated table. Or, use LOG WHERE for the weeks that contain days in both March and February/April, and manually copy the rows into the table.



RESTORE Partition Example

restore6_ppi.arc

```
LOGON dbc/sysdba,dbapass;
RESTORE DATA TABLES
  (TFACT.Sales_PPI) (ALL PARTITIONS), ABORT, RELEASE LOCK, FILE = arc6_PPI;
LOGOFF;
```

arcmain < restore6_ppi.arc

Portion of output from executing above script

```
RESTORE DATA TABLES
  (TFACT.Sales_PPI) (ALL PARTITIONS), ABORT, RELEASE LOCK, FILE = arc6_PPI;
UTILITY EVENT NUMBER - 38
LOGGED ON 4 SESSIONS
STARTING TO RESTORE TABLE "TFACT"."Sales_PPI"
Archive Bounding Condition:
RANGE_N("TFACT"."SALES_PPI"."sales_date" BETWEEN DATE '2000-01-01' AND DATE '2006-12-31' EACH INTERVAL
'1' MONTH ) IN (73 TO 75 )
[Bounding condition is well-defined]

Restore Bounding Condition:
RANGE_N("TFACT"."SALES_PPI"."sales_date" BETWEEN DATE '2000-01-01' AND DATE '2006-12-31' EACH INTERVAL
'1' MONTH ) IN (73 TO 75 )
[Bounding condition is well-defined]
DICTIONARY RESTORE COMPLETED
"Sales_PPI" - 2,391,241 BYTES, 40,500 ROWS RESTORED
"TFACT"."SALES_PPI" - LOCK RELEASED
```

COPY Statement

Use the COPY statement to recreate tables and/or databases that have been dropped or to restore them to the same system or to a different system.

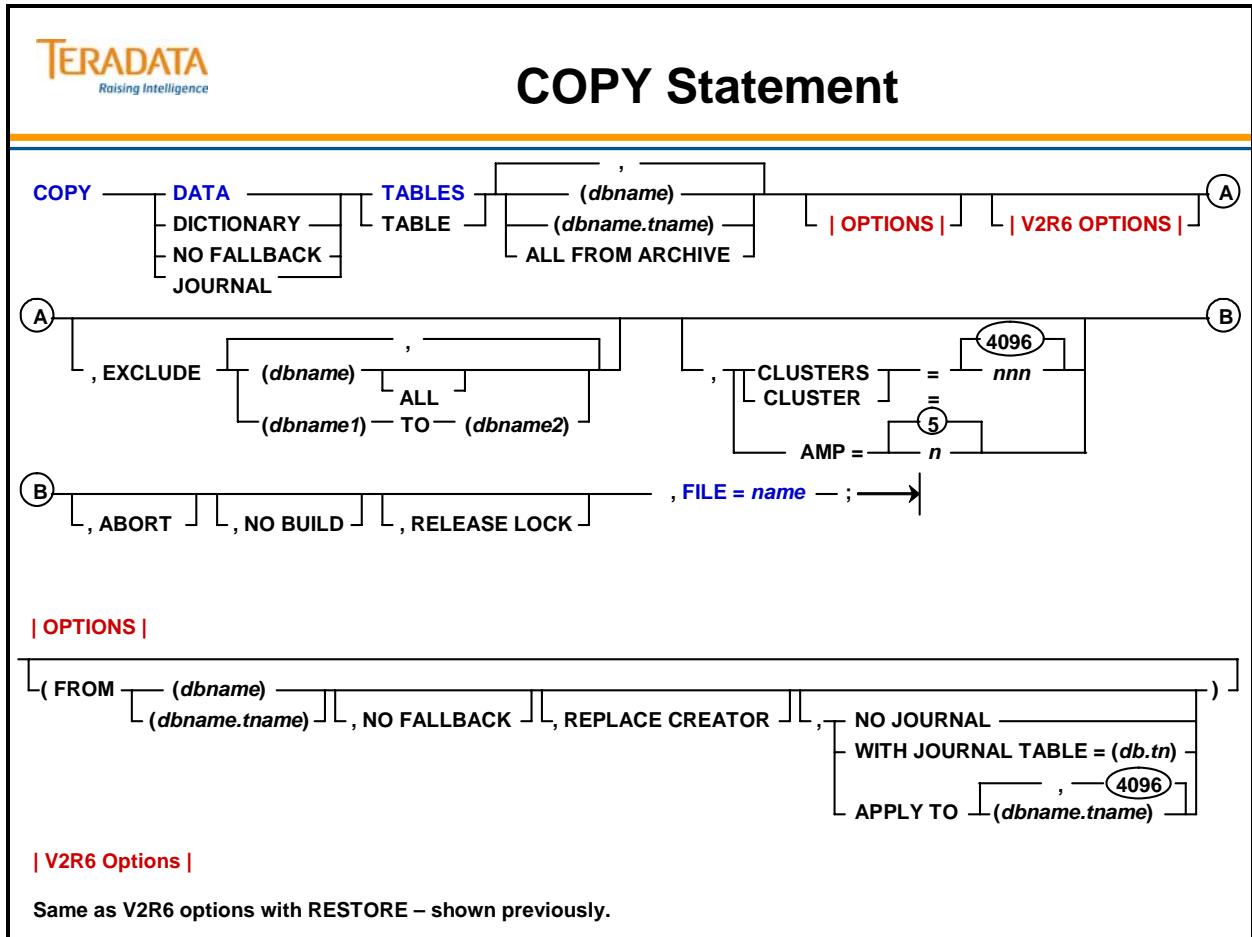
Some of the options for the COPY statement are:

NO FALBACK	Copies fallback tables into non-fallback tables. This option applies during the COPY of an all-AMP or DICTIONARY archive.
NO JOURNAL	Copies all tables with journaling disabled. This option applies during the COPY of an all-AMP or DICTIONARY archive.
WITH JOURNAL TABLE =	Overrides the default journal table of the receiving database for tables that had journaling enabled. This option applies during the COPY of an all-AMP or DICTIONARY archive.
APPLY TO	Specifies to which tables in the receiving system change images apply. This option is required when copying journal images.
NO BUILD	Prevents secondary indexes on non-fallback tables from being copied or built. On fallback tables, it prevents the creation of secondary indexes as well as fallback table rows. There is no rehashing of V1 to V2 data.
ABORT	Aborts ALL AMP copies with error messages if an AMP is offline and the restore includes a non-fallback table.
RELEASE LOCK	Causes ARC to release utility locks when a copy completes successfully.

Copying Partitioned Data

Beginning with Teradata Database V2R6.0, you can copy selected partitions of PPI tables, meaning that you can backup of one or more partitions of a table so you can archive, restore, and copy only a subset of data in a table.

To COPY selected partitions, a table must already exist on the target system. For a COPY operation, the existing table must have been created by a full-table COPY from the source machine.



Copying Objects

The COPY statement has two uses:

- It uses an archived file to recreate tables and/or databases that have been dropped.
- It copies archived files to a different system.

The COPY statement can perform one of the following tasks:

- Copy an object that has been dropped back into the original system.
- Copy an object from one system to another.
- Copy an object back to the same system.

Keyword Options with COPY

NO FALBACK Keywords

This option applies only during a copy of a dictionary archive or an all-AMPs archive. If a fallback table has permanent journaling on the archive, the table has dual journaling of its non-fallback rows after the copy when Teradata ARC applies the NO FALBACK option (unless NO JOURNAL is specified).

FROM Keyword

The object specified in the FROM keyword identifies the archive object. This option applies only during a copy of a dictionary archive or an all-AMPs archive.

Journal enabled tables in the original archive carry their journaling forward to the copy unless you specify the NO JOURNAL keywords.

The NO JOURNAL keywords apply to all tables in a database when you copy a database. This option has no effect on a receiving database's journaling defaults.

If the object you specify in the FROM option is a table, ARC copies only that table.

WITH JOURNAL TABLE Keywords

This option only applies during a copy of a dictionary archive or an all-AMPs archive. To use this option, you must have INSERT access rights to the referenced journal table. The source table must have a journal or this option has no effect.

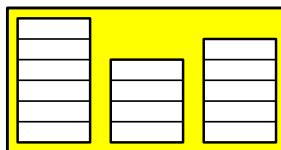
If you are copying a database, the journaling you specify with this option applies only to those tables that had journaling enabled in the original database. This option has no effect on a receiving database's default journaling.

If the database has default journaling specified, then ARC uses those options. This option only overrides the journal table in the *receiving* database, and is only valid if the originating table had journaling enabled.

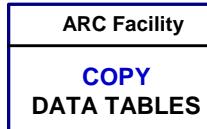
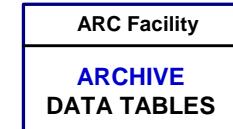


Copying Objects

Original Data Tables



Receiving Data Tables



copy1_demo.arc

```
LOGON dbc/sysdba,dbapass;
COPY DATA TABLES (SANDBOX)
  (FROM (DEMO))
, ABORT , RELEASE LOCK
, FILE = arc_demo;
LOGOFF;
```

Copy objects from archive of Demo database to Sandbox database.

copy2_demo.arc

```
LOGON dbc/sysdba,dbapass;
COPY DATA TABLE (SANDBOX.Employee)
  (FROM (DEMO),
   NO FALBACK, REPLACE CREATOR)
, ABORT , RELEASE LOCK
, FILE = arc_demo;
LOGOFF;
```

Copy Demo.Employee from archive to Sandbox.Employee and change some attributes.

Copying

Examples of the output from the previous COPY operations are shown on the facing page.

Views, Macros, Triggers, and Stored Procedures

Copying a full database archive is the same as a restore operation. ARC deletes or drops all existing tables, views, macros, stored procedures, and triggers in the receiving system.

Copying a full database archive copies all views, macros, and stored procedures in the archive to the receiving database.

But triggers cannot be copied with the COPY statement. If a trigger is defined in a database, then <trigger> NOT COPIED is displayed when the COPY statement is executed. This is not an error or warning. Triggers must be manually recreated via SQL.

You cannot copy one or more stored procedures from one database to another using the COPY statement. They can only be copied as part of a full database.

If your views, stored procedures, and macros have embedded references to databases and objects that are not in the receiving environment, those views, stored procedures, and macros will not work. To make any such views, stored procedures, and macros work, recreate or copy the references to which they refer into the receiving database.

If your views, stored procedures, and macros have embedded references to databases and objects that *are* in the receiving environment, they will work correctly.

Note: Make sure you fully qualify all table, stored procedure, and view names in a macro and all table names in a view. If you do not, you may receive an error. When you execute a COPY statement, partial names are fully qualified to the default database name. In some cases, this may be the name of the old database.

Referential Integrity

After an all-AMPs copy, copied tables do not have referential constraints. First, referential constraints are not copied into the dictionary definition tables, database DBC.ReferencedTbls and database DBC.ReferencingTbls, for either a referenced (parent) or referencing (child) table copied into a Teradata Database. Moreover, all referential index descriptors are deleted from the archived table header before it is inserted into the copied table.

For tables that already exist (same name) in the target system, reference constraints remain. However, on any table for which the copied table is a referenced table (a parent table) or a referencing table (a child table), the RI constraint will be marked in the dictionary definition tables as inconsistent.



Output of Copying Objects

Output from 1st copy example

```

... COPY DATA TABLES (SANDBOX) (FROM (DEMO))
... , ABORT
... , RELEASE LOCK
... , FILE = arc_demo;
... UTILITY EVENT NUMBER - 19
... LOGGED ON 4 SESSIONS
... "SANDBOX"."Department" CREATED
... "SANDBOX"."Employee" CREATED
... "SANDBOX"."Emp_Phone" CREATED
... "SANDBOX"."Job" CREATED
... "SANDBOX"."Salary_Log" CREATED
... STARTING TO COPY DATABASE "SANDBOX"
... "Raise_Trig" - TRIGGER NOT COPIED
... DICTIONARY COPY COMPLETED
... "Department" - 3,446 BYTES, 60 ROWS COPIED
... "Employee" - 65,077 BYTES, 1,000 ROWS COPIED
... "Emp_Phone" - 52,504 BYTES, 2,000 ROWS COPIED
... "Job" - 2,898 BYTES, 66 ROWS COPIED
... "Salary_Log" - 626 BYTES, 1 ROWS COPIED
... "SANDBOX" - LOCK RELEASED

```

Output from 2nd copy example

```

... COPY DATA TABLE (SANDBOX.Employee)
... (FROM (DEMO), NO FALBACK, REPLACE CREATOR)
... , ABORT
... , RELEASE LOCK
... , FILE = arc_demo;
... LOGGED ON 4 SESSIONS
... UTILITY EVENT NUMBER - 20
... *** Warning 3803:Table 'Employee' already exists.
... STARTING TO COPY TABLE "SANDBOX"."Employee"
... DICTIONARY COPY COMPLETED
... "EMPLOYEE" - 65,077 BYTES, 1,000 ROWS COPIED
... "SANDBOX"."EMPLOYEE" - LOCK RELEASED
...

```

BUILD Statement

The BUILD statement recreates unique and non-unique secondary indexes on non-fallback and fallback tables. This statement also builds fallback rows for fallback tables when the restore statement was performed with the NO BUILD option 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 vproc is offline during a dump or restore.
- The restore operation is not an all-AMP vproc restore.
- The archive did not include the INDEXES option.
- The restore included the NO BUILD option.

Format

DATA TABLES

JOURNAL TABLES

NO FALBACK TABLES or NO FALBACK TABLE

- This identifies the type of table to build.
- The default is NO FALBACK TABLE.
- Specify DATA TABLES when building fallback, non-fallback, or both types of tables from all AMPs. This option normally follows the restore of a cluster archive.
- Specify NO FALBACK TABLE only when building indexes for non-fallback tables.

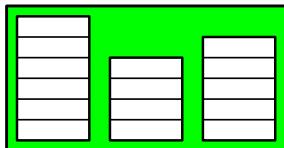
The format of the BUILD statement is shown on the facing page. The following example builds unique and non-unique secondary indexes for all tables in Sysdba and any child user/databases. The release lock option removes the utility lock after successful completion of the build operation.

BUILD DATA TABLES (Sysdba) ALL, RELEASE LOCK;



BUILD Statement

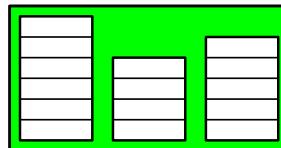
NO FALBACK Data Tables



ARC Facility
BUILD
DATA TABLES

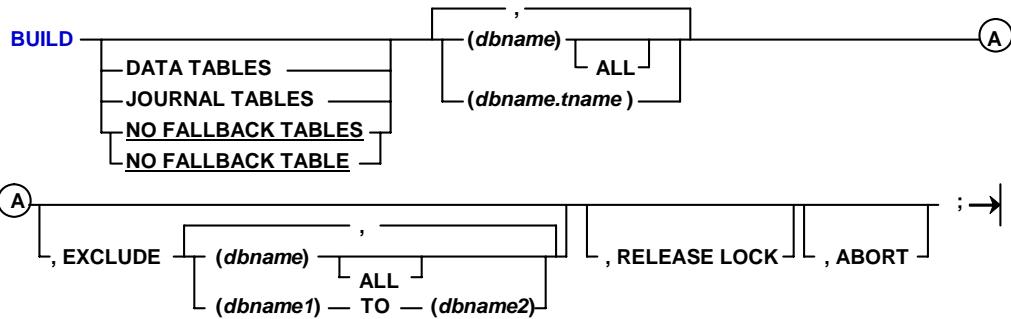
- Recreates unique secondary indexes
- Recreates non-unique secondary indexes

FALBACK Data Tables



ARC Facility
BUILD
DATA TABLES

- Recreates unique secondary indexes
- Recreates non-unique secondary indexes
- Builds fallback rows



RELEASE LOCK Statement

The ARC utility places locks on database objects while it performs archive and restore activities. These locks are referred to as 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 vproc 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 DUMP or the RESTORE privilege on the locked object. You can also release a utility-level lock if you are the owner of the locked object.

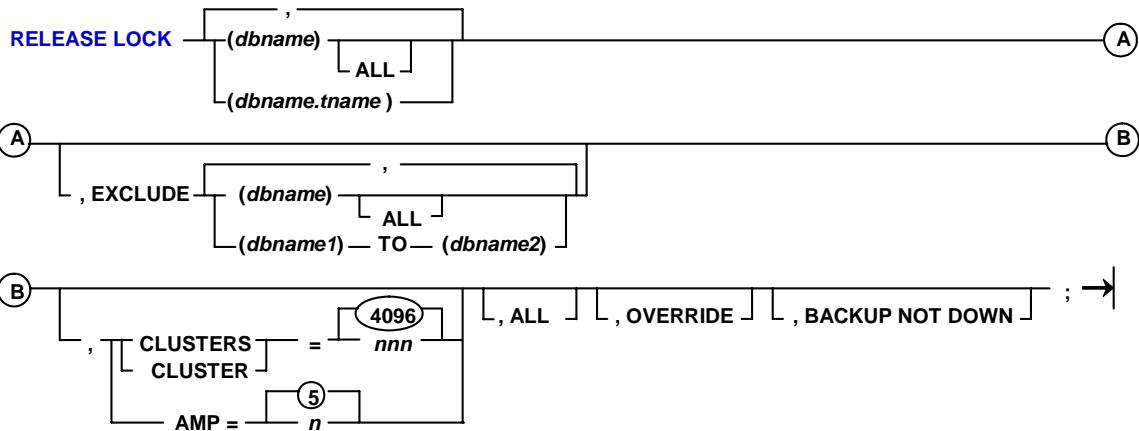
You may submit the RELEASE LOCK option at the same time you issue ARCHIVE, ROLLBACK, ROLLFORWARD, RESTORE, COPY, and BUILD commands. This accomplishes the same purpose as issuing the RELEASE LOCK statement.

The release lock syntax is shown on the facing page. Options are described below:

ALL	Releases locks on AMP vprocs that are offline when the RELEASE LOCK statement is issued. The utility releases locks when the AMP vprocs return to online operation.
OVERRIDE	Allows locks to be released by a user other than the one who set them. This option requires that the User has the DROP DATABASE privilege on the object or is an owner.
BACKUP NOT DOWN	Allows locks to remain on non-fallback tables (with single after-image journaling) for those AMP vprocs where the permanent journal backup AMP vproc is offline. The utility releases all other locks requested.



RELEASE LOCK Statement



RELEASE LOCK	You must have ARCHIVE or RESTORE privilege on the object or be the owner.
OVERRIDE	You must have DROP DATABASE privilege on the object or be an owner.
ALL	Also releases locks on offline AMPs. (Locks released when vproc is returned to service.)
BACKUP NOT DOWN	Allows locks to remain on no fallback table (with single after image journals) on vproc whose permanent journal backup vproc is down.

Revalidate References

When either referenced (parent) or referencing (child) table is restored, the reference is marked inconsistent in the database dictionary definitions. As a result, the system does not allow application users to execute 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 REVALIDATE REFERENCES FOR statement:

- Validates the inconsistent reference index on the parent table and the child table.
- Creates an error table.
- Inserts rows that fail the referential constraint specified by the reference index into the error table.

If inconsistent restraints remain after you execute the statement, you can use the statement, ALTER TABLE DROP INCONSISTENT REFERENCES, to remove them.

Required Privileges

To use the REVALIDATE REFERENCES FOR statement, the username you have specified in the LOGON statement must have one of the following privileges:

- RESTORE privileges on the table you are revalidating
- Ownership privileges on the database or table

Example

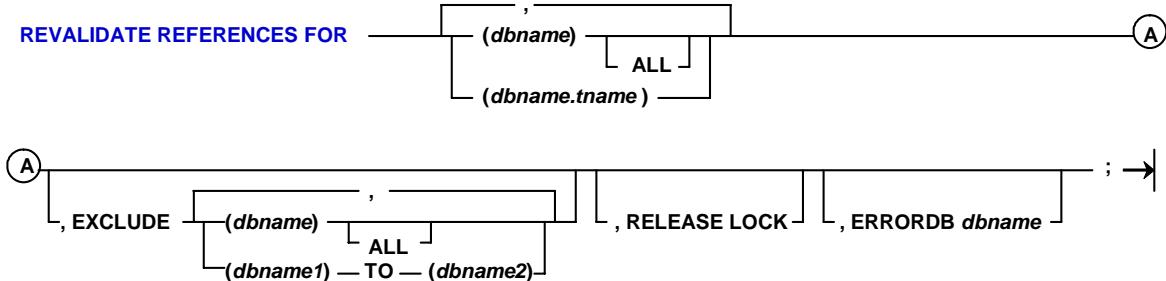
The facing page shows the syntax for the REVALIDATE REFERENCES FOR statement.



Revalidate References

The REVALIDATE REFERENCES FOR statement validates the inconsistencies between foreign and parent keys.

This allows users to execute UPDATE, INSERT and DELETE statements on the tables.



Example:

revalidate_ref_pd.arc

```

LOGON dbc/sysdba,dbapass;
REVALIDATE REFERENCES FOR (PD) , RELEASE LOCK;
LOGOFF;

```

Revalidate references
for the PD database.

Revalidate References Output

The facing page illustrates the output from using the REVALIDATE REFERENCES FOR statement in the previous example.

Note: Error tables were not created for Department and Emp_Phone because all of the foreign key values are parent key values. The Revalidate References command will only create “error tables” if there are invalid foreign key values.



Revalidate References Output

revalidate_ref_pd.arc

```
LOGON dbc/sysdba,dbapass;
REVALIDATE REFERENCES FOR (PD)
, RELEASE LOCK;
LOGOFF;
```

Output from this restore script

```
:
... REVALIDATE REFERENCES FOR (PD)
... , RELEASE LOCK;

... VALIDATING REFERENCE INDEX '0' FOR TABLE PD.DEPARTMENT
... REFERENCES VALIDATED FOR INDEX 0.
... NO ERRORS FOUND.

... VALIDATING REFERENCE INDEX '0' FOR TABLE PD.EMPLOYEE
... REFERENCES VALIDATED FOR INDEX 0.
... 1 ERRORS FOUND AND LOGGED IN TABLE PD.EMPLOYEE_0

... VALIDATING REFERENCE INDEX '4' FOR TABLE PD.EMPLOYEE
... REFERENCES VALIDATED FOR INDEX 4.
... 1 ERRORS FOUND AND LOGGED IN TABLE PD.EMPLOYEE_4

... VALIDATING REFERENCE INDEX '8' FOR TABLE PD.EMPLOYEE
... REFERENCES VALIDATED FOR INDEX 8.
... 1 ERRORS FOUND AND LOGGED IN TABLE PD.EMPLOYEE_8

... VALIDATING REFERENCE INDEX '0' FOR TABLE PD.EMP_PHONE
... REFERENCES VALIDATED FOR INDEX 0.
... NO ERRORS FOUND.

... PD.EMPLOYEE - LOCK RELEASED
... PD.DEPARTMENT - LOCK RELEASED
... PD.JOB - LOCK RELEASED
... PD.EMP_PHONE - LOCK RELEASED
... STATEMENT COMPLETED
:
```

Recovery Control Data Dictionary Views

There are four system views that contain information about ARC utility events. The name, purpose, and dictionary table name of each view is listed below.

DBC.Association[X] Views

Provides information about objects that have been imported from another Teradata Database system or otherwise created using the ARC COPY statement. Table name: DBC.RCEvent.

DBC.Events[X] Views

Provides a row for each archive and recovery activity. Table name: DBC.RCEvent.

DBC.Events_Configuration[X] Views

Provides information about archive and recovery activities that did NOT affect all AMP vprocs. Table name: DBC.RCConfiguration.

DBC.Events_Media[X] Views

Provides information about archive and recovery activities that involved removable media. Table name: DBC.RCMedia.



Recovery Control Data Dictionary Views

<u>View Name</u>	<u>Description</u>
DBC.Association[X]	Provides information about objects you import from another database system (an example is provided).
DBC.Events[X]	Provides an audit trail of all archive and recovery activity (an example is provided).
DBC.Events_Configuration[X]	Provides information about archive and recovery activities that did not affect ALL AMPs.
DBC.Events_Media[X]	Provides information about archive and recovery activities that involve removable media.

Association View

The DBC.Association[X] views allow you to retrieve information about an object imported from another Teradata Database.

An existing object created with the ARC utility COPY statement also displays 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.

Example

The example on the facing page uses the Association view to list all tables, views, or macros that were copied into the Sandbox database. The result of the query displays imported table names. The object column displays the current name of each table. The “From_Source” column provides the name of the original table. The event column shows the event number assigned to the copy operation.



Association View

Provides information about COPY operations.

Enables you to retrieve information about an object imported from another Teradata database.

DBC.Association[X]

DatabaseName*	TableName	EventNum
Original_DatabaseName	Original_TableName	
Original_TableKind	Original_Version	
Original_ProtectionType	Original_JournalFlag	
Original_CreatorName	Original_CommentString	

* DatabaseName: The name of the database or user where the imported object now resides.

Example: List all objects copied into the Sandbox database.

```
SELECT TRIM(DatabaseName) || '.' || TableName (FORMAT 'X(25)') AS Object
      ,TRIM(Original_DatabaseName) || '.' || Original_TableName (FORMAT 'X(25)') AS From_Source
      ,EventNum (FORMAT 'Z(4)9') AS Event
FROM DBC.Association
WHERE DatabaseName LIKE '%Sandbox%'
ORDER BY Event_Num, Object;
```

Object	From_Source	Event
Sandbox.Department	DEMO.Department	19
Sandbox.Emp_Phone	DEMO.Emp_Phone	19
Sandbox.Job	DEMO.Job	19
Sandbox.Salary_Log	DEMO.Salary_Log	19
Sandbox.Employee	DEMO.Employee	20

Events View

The DBC.Events[X] views track ARC activity. The ARC utility inserts a new row in the Events system table each time another ARC activity begins. The Events views return a row for each activity tracked. Each event type is listed below:

Checkpoint Event Row	Created for each journal checkpointed
Copy Event Row	Created for each database or table copied
Delete Event Row	Created for each journal deleted
Dump Event Row	Created for each database or table dumped
Restore Event Row	Created for each database or table restored
Rollback Event Row	Created for each database or table rolled back
Rollforward Event Row	Created for each database or table rolled forward

Example

The SQL statement on the next page requests a list of all ARC activity that took place on January 13, 2006. The results display five ARC activities.



Events View

Provides an audit trail of all archive and recovery activities for objects visible to you.

DBC.Events[X]

CreateDate	AllAMPsFlag	LockMode	CreateTime
RestartSeqNum	JournalUsed	EventNum	OperationInProcess
JournalSaved	EventType	TableName	IndexPresent
UserName	CheckpointName	DupeDumpSet	DatabaseName
LinkingEventNum*	ObjectType	DataSetName	

Example:

List all ARC activity that occurred on Jan 13, 2006.

```
SELECT CreateDate
      ,EventNum      (FORMAT 'Z(4)9') AS Event
      ,UserName     (FORMAT 'X(12)')
      ,EventType    (FORMAT 'X(12)')
      ,DatabaseName (FORMAT 'X(12}') AS DBName
   FROM DBC.Events
 WHERE CreateDate = '2006-01-13'
 ORDER BY EventNum ;
```

Example Results:

CreateDate	Event	UserName	EventType	DBName
2006-01-13	15	SYSDBA	Dump	PD
2006-01-13	16	SYSDBA	Dump	PD
2006-01-13	17	SYSDBA	Restore	PD
2006-01-13	18	SYSDBA	Restore	PD
2006-01-13	19	SYSDBA	Copy	SANDBOX
2006-01-13	20	SYSDBA	Copy	SANDBOX

Restoring Data Summary

The facing page summarizes some important concepts regarding this module.



Summary

- Restore operations transfer database information from archive files stored on portable media to all AMP vprocs, AMP clusters or specified AMP vprocs.
- You can restore archived data tables to the database if the data dictionary contains a definition of the entity you wish to restore.
- The primary statements that you use in recovery operations are:
 - ANALYZE
 - REVALIDATE REFERENCES FOR
 - RESTORE
 - RELEASE LOCK
 - COPY
 - BUILD
- Teradata features several recovery control system views that contain information about ARC utility events.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

- 1. True or False.** You can use the RESTORE command to restore entities that are not defined in the data dictionary.
- 2. True or False.** When you execute a RESTORE of a database, any tables or views created since the archive of the database are dropped when you restore the database.
- 3. True or False.** You can use the COPY operation to copy tables, views, macros, and triggers from one system to another system.
- 4. The REVALIDATE REFERENCES FOR statement is used to validate Referential Integrity between tables that are identified as _____.**
 - A. Inconsistent
 - B. Unresolved
 - C. Missing
 - D. Invalid

Teradata Training

Notes

Module 60



Data Recovery Operations

After completing this module, you will be able to:

- **Describe how to use the following statements to recover archived data back to the Teradata Database:**
 - CHECKPOINT
 - DELETE JOURNAL
 - ROLLBACK
 - ROLLFORWARD

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Data Recovery Using Roll Operations	60-4
The CHECKPOINT Statement	60-6
CHECKPOINT WITH SAVE Statement.....	60-8
Example	60-8
Checkpoint Lock Mechanisms	60-8
Checkpoint with Offline AMPs	60-8
Using the ROLLBACK Command	60-10
Example	60-10
NO DELETE Option.....	60-10
The ROLLBACK Statement.....	60-12
ROLLFORWARD Statement	60-14
Example	60-14
PRIMARY DATA Option	60-14
ROLLFORWARD Restrictions	60-16
AMP-Specific Restore	60-16
All-AMP Restore	60-16
Example	60-16
The ROLLFORWARD Statement.....	60-18
DELETE JOURNAL Statement	60-20
Access Privileges	60-20
Restrictions.....	60-20
Summary	60-22
Review Questions	60-24

Data Recovery Using Roll Operations

The restore statement allows you to move information from archive files back to the Teradata database. The restore operation can restore data or journal tables.

After you execute a RESTORE statement, data tables are ready to use.

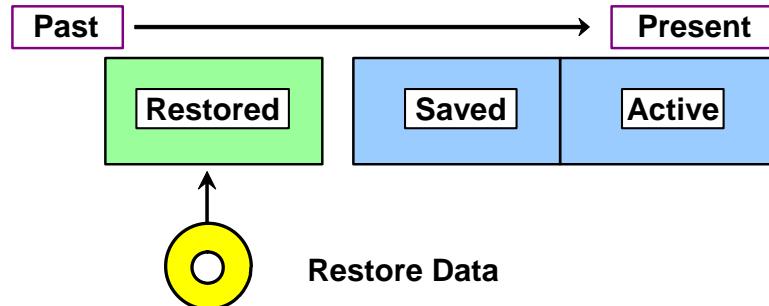
When you restore a journal table, the system restores the information to a permanent journal subtable. Before you can use the tables, you must perform a rollback or rollforward operation to move the journal tables back to the data tables.

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.

Data Recovery Using Roll Operations



- The RESTORE function copies journal archive files to the restored subtable of the permanent journal.
- ROLLBACK and ROLLFORWARD statements apply journal table contents to data tables.
- Roll operations can use:
 - Current journal (active and saved subtable)
 - Restored journal (restored subtable)

The 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. The 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.

Use the following options with the CHECKPOINT statement:

WITH SAVE

Use this option before you archive saved journal images to a host media to append the active journal subtable to the saved journal 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 saved journal subtable has no fixed size and can grow to the limit of the database.

**USE ACCESS
LOCK**

A checkpoint with save may optionally use an access lock. Without this option, the system must acquire a read lock on all tables assigned to the journal being checkpointed.

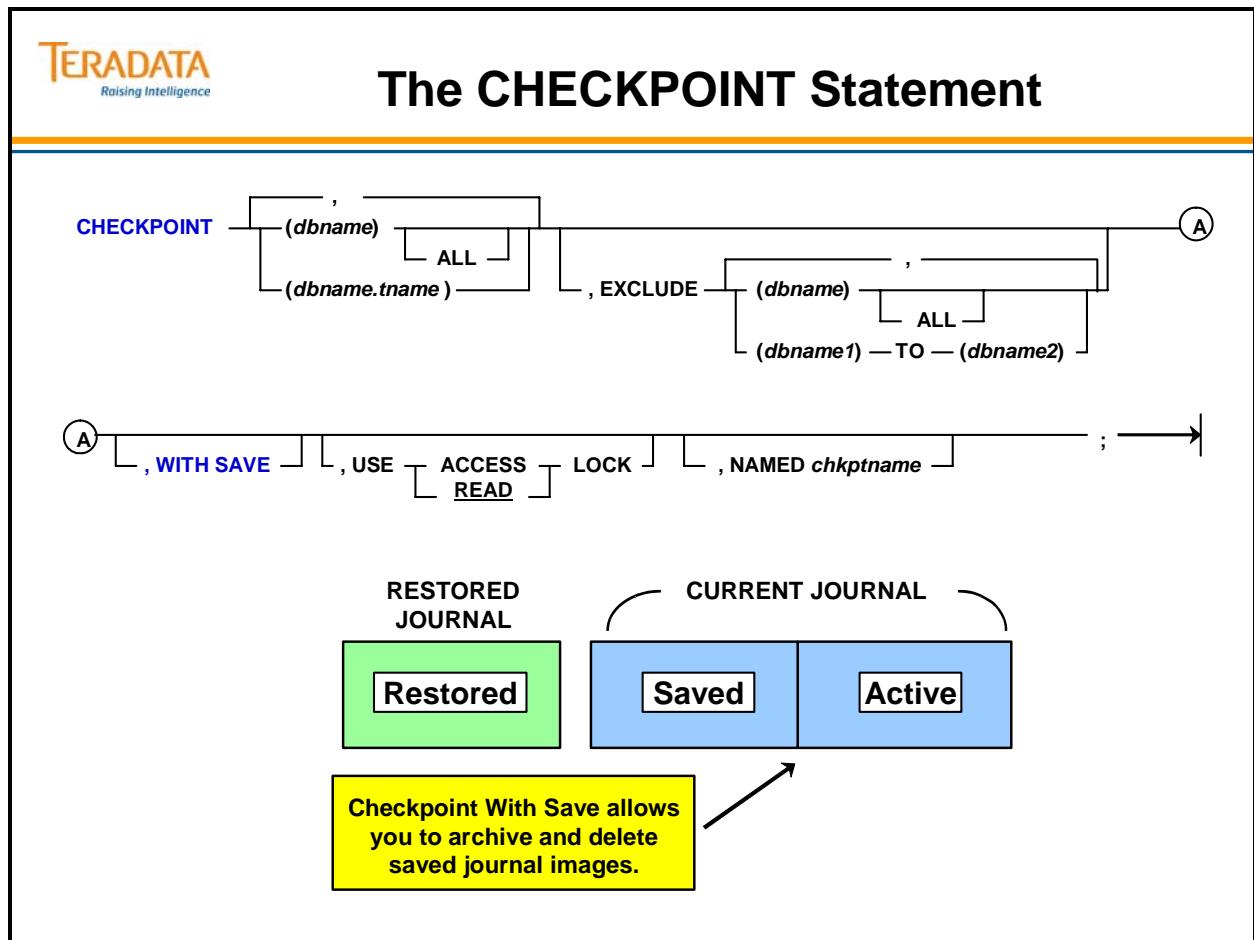
Updates to tables may continue when you use the access lock option. Updates in progress complete after the checkpoint.

**NAMED
checkpointname**

Checkpoint names may be up to 30 characters long and are not case-specific. Teradata software 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.



CHECKPOINT WITH SAVE Statement

The CHECKPOINT WITH SAVE option inserts a marker row and appends any stored images preceding the marker row from the active to the saved subtable. The database automatically initiates a new active subtable. You can dump the contents of the saved subtable to an archive file.

Example

The facing page shows two different current journals, before and after a checkpoint operation. The active subtable before checkpoint contains five change image rows. After checkpoint with save, the active subtable is empty, and the saved subtable contains the five change rows and a marker row.

Checkpoint Lock Mechanisms

The default lock mechanism for the checkpoint command is read 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 USE LOCK option permits users to assign an access lock rather than a read lock. 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.

Since users cannot know which side of a checkpoint a particular transaction will fall on, restoring to a checkpoint created under an access lock cannot guarantee that transactions in progress at the time of the checkpoint will be included in that restore.

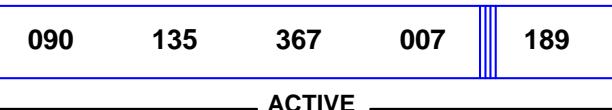
Checkpoint with Offline AMPs

An individual AMP may be off-line when you issue the checkpoint command. In this case, the utility automatically generates a system log entry that marks the checkpoint as soon as the AMP comes back on-line. The system startup process generates the checkpoint and requires no user input.



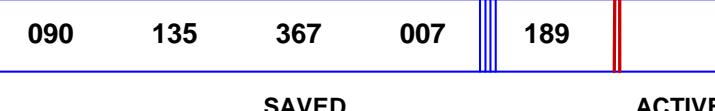
CHECKPOINT WITH SAVE Statement

CURRENT JOURNAL



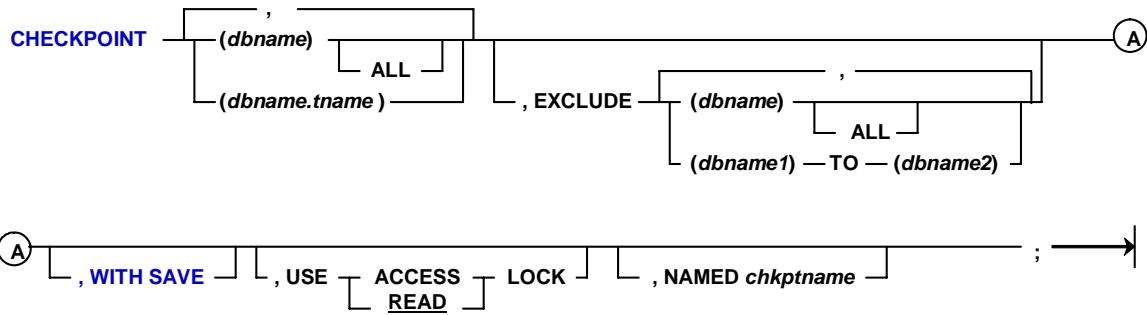
**CHECKPOINT Payroll_Tab.Salaries_Jnl
, WITH SAVE ;**

CURRENT JOURNAL



ACTIVE

"Saved CHECKPOINT"



Using the ROLLBACK Command

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 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. The Teradata database does not have any simple tools for looking at journal subtables to determine that they contain the desired data.

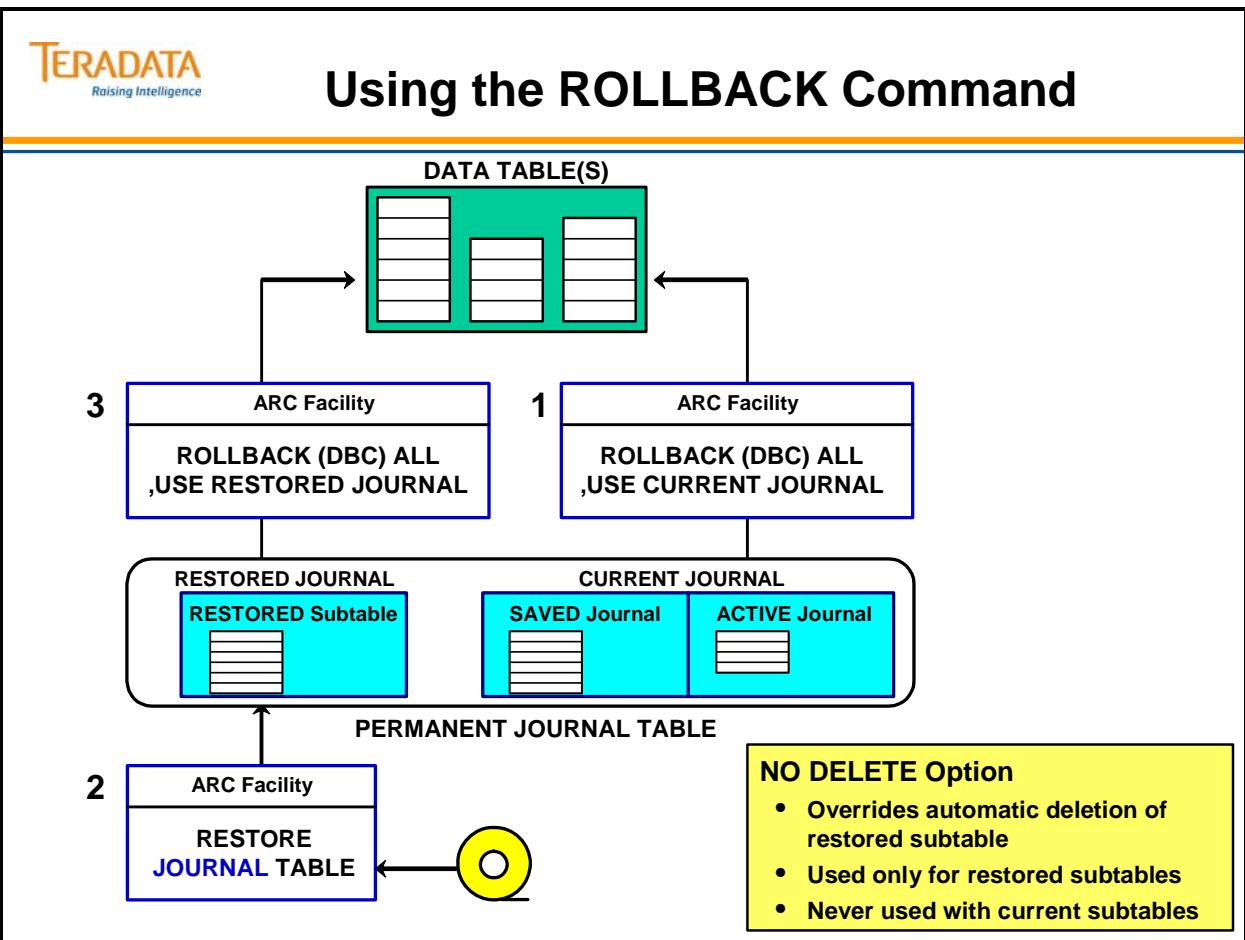
Example

The example on the facing page illustrates a rollback procedure. First, (step 1), activate the ROLLBACK CURRENT JOURNAL statement to rollback any changes made since the journal table was archived. This statement rolls back the current subtable. Next (step 2), run the RESTORE JOURNAL TABLE command to load the appropriate archive file into the restored subtable of the permanent journal.

Finally (step 3), 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

NO DELETE Option

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, enables you to recover selected tables first, and then later recovers other tables that may have changes in the journal.



The ROLLBACK Statement

To recover from one or more transaction errors, use the ROLLBACK statement. 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.

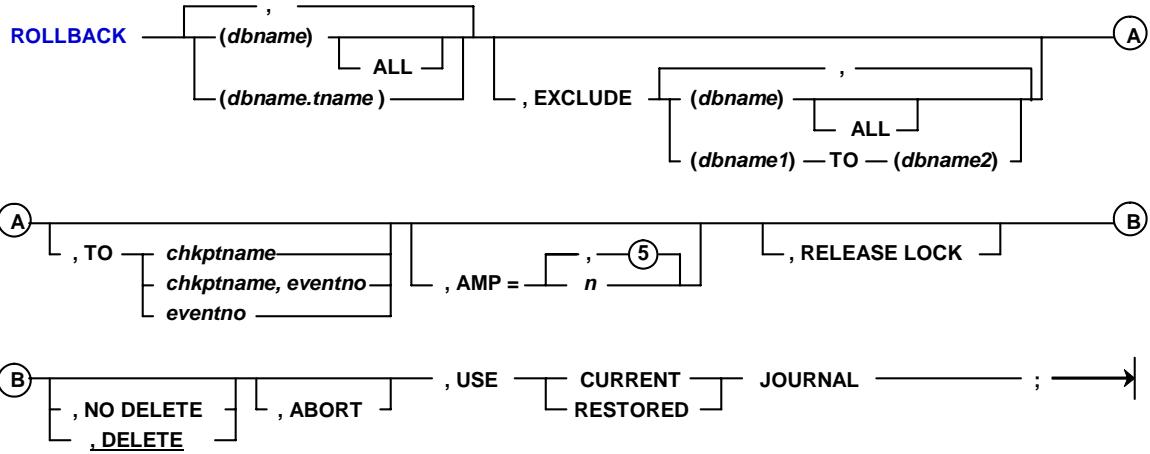
The facing page shows the format of the ROLLBACK statement. A description of the “TO CHECKPOINT” option follows:

TO checkpointname, eventno 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 one of these or both. To find the checkpoint names or event numbers, select information about the checkpoint from the DBC.Events view.

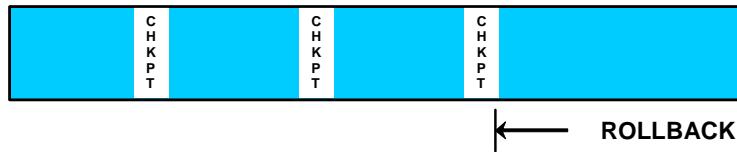
If there are duplicate checkpoint names in the journal and an event number is not supplied, rollback stops at the last chronological entry made with a matching name.



The ROLLBACK Statement



Use ROLLBACK to recover from a transaction error.



ROLLFORWARD Statement

The ROLLFORWARD command helps you recover from a hardware error and 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 process ensures that you restore the correct subtable.

Example

The example on the facing page illustrates a rollforward procedure. First, the administrator runs the RESTORE DATA TABLE command. Then, he/she runs the RESTORE JOURNAL TABLE command to load the appropriate archive files into the restored permanent journal subtable. Next, he/she submits the ROLLFORWARD RESTORED JOURNAL command to replace existing data table rows with their after-image rows stored in the restored journal.

Lastly, he/she activates 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

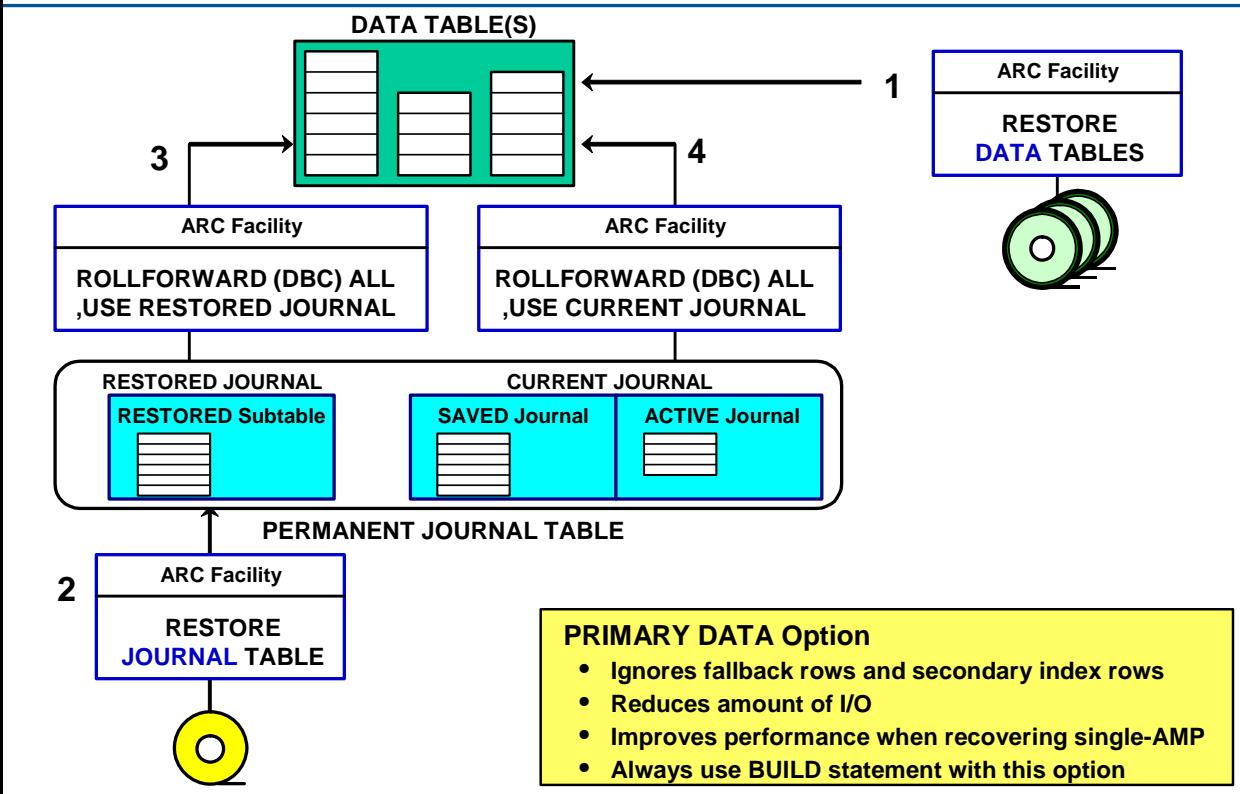
This option replaces only primary row images during the rollforward process. It ignores secondary index and fallback rows.

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.



Using the ROLLFORWARD Command



ROLLFORWARD Restrictions

The diagrams on the facing page illustrate several important restrictions in using the ROLLFORWARD statement.

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.

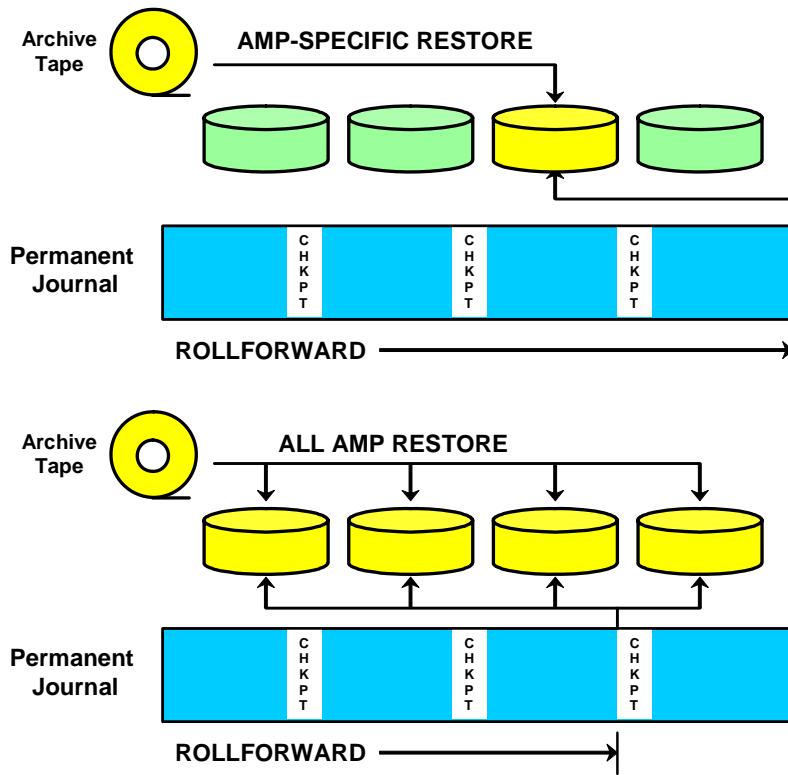
Note: Use the DBC.Events view to determine event numbers and/or checkpoint names.

Example

```
SELECT EventNum FROM DBC.Events  
  WHERE CreateDate = '2005-02-15';  
  
SELECT CheckPointName FROM DBC.Events  
  WHERE CreateDate = '2005-02-15';
```



ROLLFORWARD Restrictions



Following an AMP-specific restore, a ROLLFORWARD is permitted only to the end of the journal.

After an ALL-AMP restore, a ROLLFORWARD may be done to a checkpoint, or to the end of the journal.

The ROLLFORWARD Statement

Use the ROLLFORWARD statement to recover from a hardware error. Before you can rollforward, you must have a backup copy of the table rows and AFTER Image journal rows since the last backup.

The format of the ROLLFORWARD statement is shown on the next page. A description of some of the options follows:

PRIMARY DATA

During a rollforward operation, this option instructs the software to ignore secondary index and fallback row updates. A BUILD operation will rebuild the invalidated fallback copy and indexes.

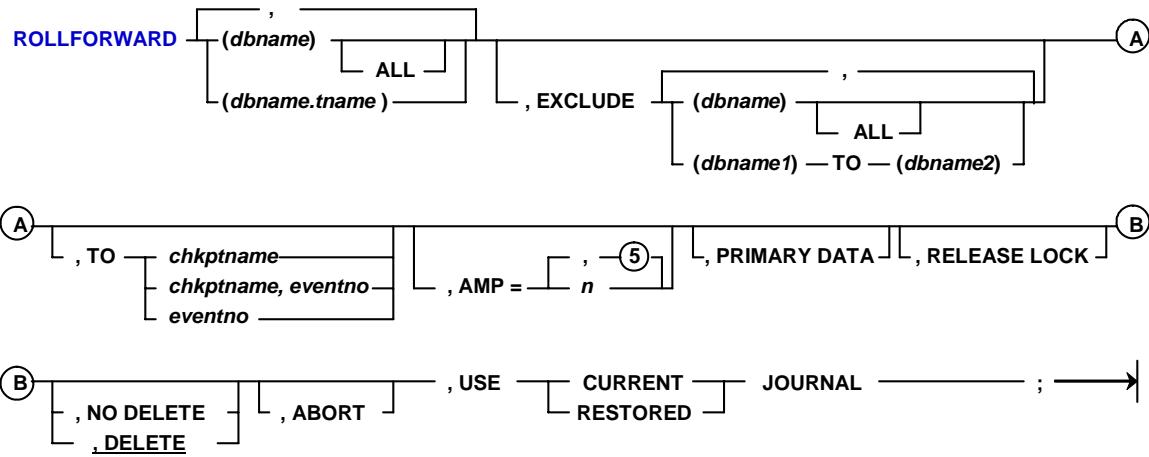
TO checkpointname, eventno

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.

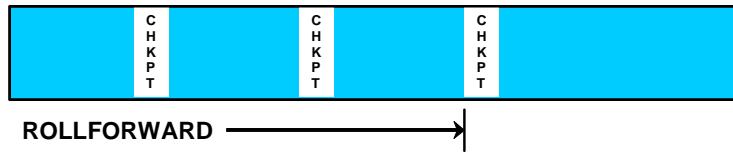
If there are duplicate checkpoint names in the journal and an event number is not supplied, rollforward stops when it encounters the first chronological entry made with a matching name.



The ROLLFORWARD Statement



Use ROLLFORWARD to recover from a hardware failure.



DELETE JOURNAL Statement

The DELETE JOURNAL command enables you to erase the contents of either the restored subtable or the saved subtable of a permanent journal. You cannot delete the contents of the active subtable. You must have the RESTORE privilege to execute this command.

The facing page shows the DELETE JOURNAL statement.

Access Privileges

To delete a journal table, the user name specified in the LOGON statement must have one of the following:

- The RESTORE privilege on the database or journal table being deleted
- Ownership of the database containing the journal table

Restrictions

You cannot delete a saved subtable when all of the following conditions are true:

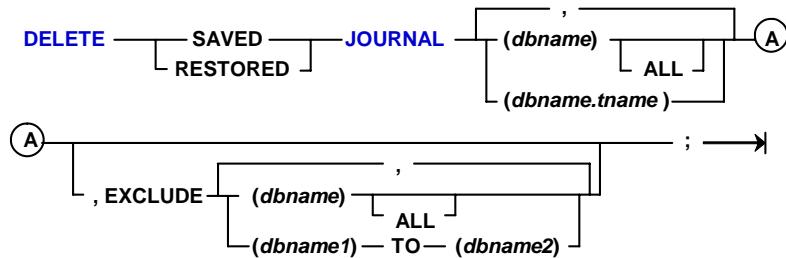
- A CHECKPOINT statement in the archive utilized an access lock, and
- The journal is not dual image, and
- One or more AMPs are off-line.

Transactions between an all-AMP archive and a single-AMP archive may not be consistent when a journal archive has all three of the above conditions. You cannot delete a saved subtable with an AMP off-line that does not have a dual journal.

The command does not delete the rows in the active journal.



DELETE JOURNAL Statement



To use the **DELETE JOURNAL** statement, you must have the **RESTORE** privilege or own the database that contains the journal.

Note: You cannot delete rows from an active journal.

Summary

The facing page summarizes some important concepts regarding this module.



Summary

- As like archive and restore operations, you use the ARC facility for recovery operations.
- Roll operations can use either current journals (active and saved subtables) or restored journals (restored subtable).
- 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** 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 permanent journal.

Review Questions

Check your understanding of the concepts discussed in this module by completing the review questions as directed by your instructor.



Review Questions

1. **True or False.** The DELETE JOURNAL command can be used to delete the active and the saved areas of the current journal.
2. In general, rollback operations help you recover from _____ failures and rollforward operations help you recover from _____ failures.
3. To use the ARCHIVE JOURNAL TABLE command to archive a permanent journal, the active journal images need to be moved to the saved area of the current journal. The command to do this is:

Teradata Training

Notes

Module 61



Teradata Factory Recap

After completing this module, you will be able to:

- Identify those Data Dictionary tables that need periodic maintenance.
- Identify those administrative functions that are NOT necessary with Teradata.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Teradata Factory Review – Week 1	61-4
Teradata Factory Review – Week 2	61-6
Dictionary Tables to Maintain	61-8
Plan and Follow-up	61-10
Things You Never Have to do with Teradata	61-12
Things You Never Have to do with Teradata (cont.).....	61-14
Teradata Differentiators (Part 1)	61-16
Teradata Differentiators (Part 2)	61-18
Teradata Certification – Materials to Focus on.....	61-20

Teradata Factory Review – Week 1

The facing page lists some of the key topics that were covered in the first week of this course.



Teradata Factory Review – Week 1

Teradata Concepts – Big Picture view of Teradata

- Teradata Database concepts, architecture, and terminology
- Data protection mechanisms (RAID, Cliques, Clusters, Locks, and Journals)
- Teradata Systems and Configurations (e.g., Teradata 5550)
- How Teradata uses memory and utilizes disk array storage

GOAL – Understand basic concepts of Teradata and how Teradata fits with NCR systems

Physical Database Design and Implementation – Detailed view of Teradata

- Data Distribution, Hashing, and Index Access
- Analyze Primary Index Criteria and choose Primary Indexes
- Analyze Secondary Index Criteria and choose Secondary Indexes
- Access Issues, Constraints, Sizing, and Statistics
- Understand Join Processing and interpret EXPLAIN plans of joins
- Additional Index Choices – Join Indexes, Hash Indexes, etc.

GOAL – Create tables with appropriate attributes and indexes.

Teradata Factory Review – Week 2

The facing page lists some of the key topics that were covered in the second week of this course.



Teradata Factory Review – Week 2

Teradata Application Utilities – Load and Export data

- Utilization of BTEQ, FastLoad, FastExport, MultiLoad, and TPump utilities

GOAL – Create load and export scripts to load/export data into/from Teradata tables

Teradata Database Administration

- Creating and using the Database Environment and Hierarchy
- The Data Dictionary/Directory
- Space Allocation and Usage
- Management of Users and Databases
- Controlling access to system via Access Rights, Roles, and Profiles
- Monitoring system activity and logging user access and queries
- Workload Management (TDWM and TASM)
- Utilization of tools such as Teradata Administrator and Teradata Manager
- System utilities – administrative, maintenance, and recovery

GOAL – Administration of a Teradata Database using DBA commands and utilities

Dictionary Tables to Maintain

You need to maintain some dictionary tables. The following pages list these tables and describe the maintenance.



Dictionary Tables to Maintain

Reset accumulators and peak values using DBC.AMPUsage view and the ClearPeakDisk macro provided with the software.

DBC.Acctg
Resource usage by Account/user

DBC.DataBaseSpace
Dbase and Table space accounting

Teradata automatically maintains these tables, but good practices can reduce their size.

DBC.AccessRights
User Rights on objects

DBC.RoleGrants
Role rights on objects

DBC.Roles
Defined Roles

DBC.Accounts
Account Codes by user

Archive these logging tables (if desired) and purge information 60-90 days old. Retention depends on customer requirements.

DBC.SW_Event_Log
Database Console Log

DBC.ResUsage
Resource monitor tables

DBC.EventLog
Session logon/logoff history

DBC.AccLogTbl
Logged User-Object events

DBC.DBQL tables
Logged user/SQL activity

Purge these tables when the associated removable media is expired and over-written

DBC.RCEvent
Archive/Recovery events

DBC.RCConfiguration
Archive/Recovery config

DBC.RCMedia
VolSerial for Archive/Recovery

Plan and Follow-up

Establish a set of procedures that will help you administer the Teradata Database. Document these procedures and periodically refer to them.



Plan and Follow-up

Review the material you have learned and experienced in this course. From it, develop a checklist of tasks. For example:

1. Set up a job that periodically checks the size of your dictionary tables.
2. Set up a job that periodically checks the size of your application databases. Evaluate them for even data distribution on AMPs. Ensure that user's permanent space is being used efficiently. Reallocate space if necessary.
3. Verify adequate Spool_Reserve.
4. Set up and document the definition of users, roles, profiles, privileges, and an accounting system (if you don't have one already).
5. Check the allocation of the Crashdumps database.
6. Install and run the ResUsage macros at regular intervals. Evaluate and review reports for even distribution of processing.

Things You Never Have to do with Teradata

The facing page lists a number of database maintenance functions that you never have to do with the Teradata database.



Things You Never Have to do with Teradata

Implementation

1. Create and format data files to hold the data and the indexes.
2. Determine the physical location of each table and index partition or simple tablespace.
3. Write programs to determine how to divide data into partitions.
4. Code the space allocation for each partition or underlying file structures.
5. Embed partitioning assignments into CREATE TABLE statements.
6. Code the definition and allocation for temporary work space.
7. Create, size, and determine the content of tablespaces.
8. Associate tables and/or queries with degrees of parallelism.
9. Add hints or otherwise rewrite SQL.
10. Determining the level of parallelism to be assigned to tables or indexes.
11. Assign and manage special buffer pools for parallel processing.
12. Create rollback segments or log files.
13. Ensure that the data is spread evenly across disks and controllers.
14. Build summary tables before end users can access the data warehouse.
15. Carefully build indexes on tables and summary tables to support index only access for performance, based on known queries – these indexes may or may not aid in ad hoc access.
16. Build and partition materialized view logs.
17. Build and partition indexes on top of materialized views.
18. Determine how materialized views are updated, asynchronously or synchronously.

Things You Never Have to do with Teradata (cont.)

The facing page lists a number of database maintenance functions that you never have to do with the Teradata database.



Things You Never Have to do with Teradata (cont.)

Support

19. Monitor partition size.
20. Monitor and tune temporary work and sort spaces.
21. Monitor and tune buffer pool assignments.
22. Monitor and tune parameters and control blocks that enable parallel execution.
23. Perform periodic table and index reorgs (unloads and reloads, dropping and rebuilding).
24. Convert data types of mainframe data sets prior to a data warehouse load.
25. Setting up multiple load jobs from a mainframe to the data warehouse in order to load a single table in parallel.
26. Manually restart the multi-step load process when failure occurs.
27. Sort and/or split the data before a load job.

Growth and Leverage

28. Alter the parallelism assignments as the number of users or data volume increases.
29. Expand partition boundaries or relocate partition data sets.
30. Add or delete table or index partitions as tables grow.

Teradata Differentiators (Part 1)

The facing page identifies a number of key Teradata differentiators.



Teradata Differentiators (Part 1)

- **Product Maturity** – Teradata has focused on data warehousing needs since its initial release in 1984.
- **Customer References** – Teradata's impressive list of customers includes leaders in their respective industries and also in the use of data warehousing technology.
- **Quickest Time to Solution** – By the nature of the inherent parallel architecture and self-managing features, Teradata provides the flexibility needed for rapid, initial implementation, and ongoing extensibility.
- **Lowest Total Cost of Ownership** – even the largest Teradata sites report having two or fewer full-time DBAs.
- **Complete Support Infrastructure** – with almost 20 years of data warehousing experience, Teradata is supported by the most skilled and experienced data warehousing professionals in the industry.
- **Effortless Scalability** – unique, unconditional parallelism and automatic hashed data distribution are the key reasons behind Teradata's scalability.

Teradata Differentiators (Part 2)

The facing page identifies additional Teradata differentiators.



Teradata Differentiators (Part 2)

- **High User Concurrency** – Teradata offers industry-leading performance to increasing numbers of satisfied users as the warehouse workload grows.
- **Complex and Ad Hoc Query Performance** – Teradata's parallel-aware, cost-based optimizer combined with its hashed based data distribution, provides the most advanced ad hoc and complex query environment in the market today.
- **Fast, Fail-safe Data Load Utilities** – Teradata ensures mission-critical availability of the business information by allowing load, update, purge, archive, and restore while users access the warehouse.
- **Seamless Mainframe Integration** – Teradata offers bi-directional, high-speed channel connectivity to leading mainframe environments. Teradata also complies with open standards for seamless interoperability with other client platform environments.

Teradata Certification – Materials to Focus on

The flow chart on the facing page contains the areas or topics that you need to focus on and study to prepare for the Teradata Certification tests.

These materials are to only be used by PS consultants as reference materials.

These materials are not to be distributed to customers in either electronic or printed formats. They are also not to be used as teaching materials by associates outside the Teradata Division Training organization without the express consent of the Teradata Division Training organization.



Teradata Certification – Materials to Focus On

Certified Teradata Professional Test 1 (Basics)

Modules 1-10

Binder 1 – Modules 14 (pg 1-25), 18 (pg 1-19), 25 (pg 46-49), 29 (pg 51-53)

Binder 2 – Modules 30 (pg 5), 31 (pg 34-45), 32 (all)

✓ This course helps prepare you for these Teradata Certification tests.



Certified Teradata Implementation Specialist Test 2

Binder 1 – Modules 10, 13 to 29 (all)

Binder 2 – Modules 30 to 32, 40 to 43 (all)

Certified Teradata SQL Specialist Test 3

Attend WBT or Customer Education SQL classes

Certified Teradata Administrator Test 4

75% – Modules 41 to 60 (all),
25% – Binders 1 & 2 ✓

Certified Teradata Designer Test 5

70% – Modules 14 to 31 (all)
30% – Binders 1 & 2 ✓

Certified Teradata Application Developer Test 6

80% – Both Sections
20% – Advanced SQL, OLAP, Preprocessor, CLI, ODBC, DATE, etc.

Teradata Training

Notes

Module A



Appendix A: The Lab Environment

This appendix describes the lab environment that will be used during the class.

Teradata Proprietary and Confidential

Teradata Training

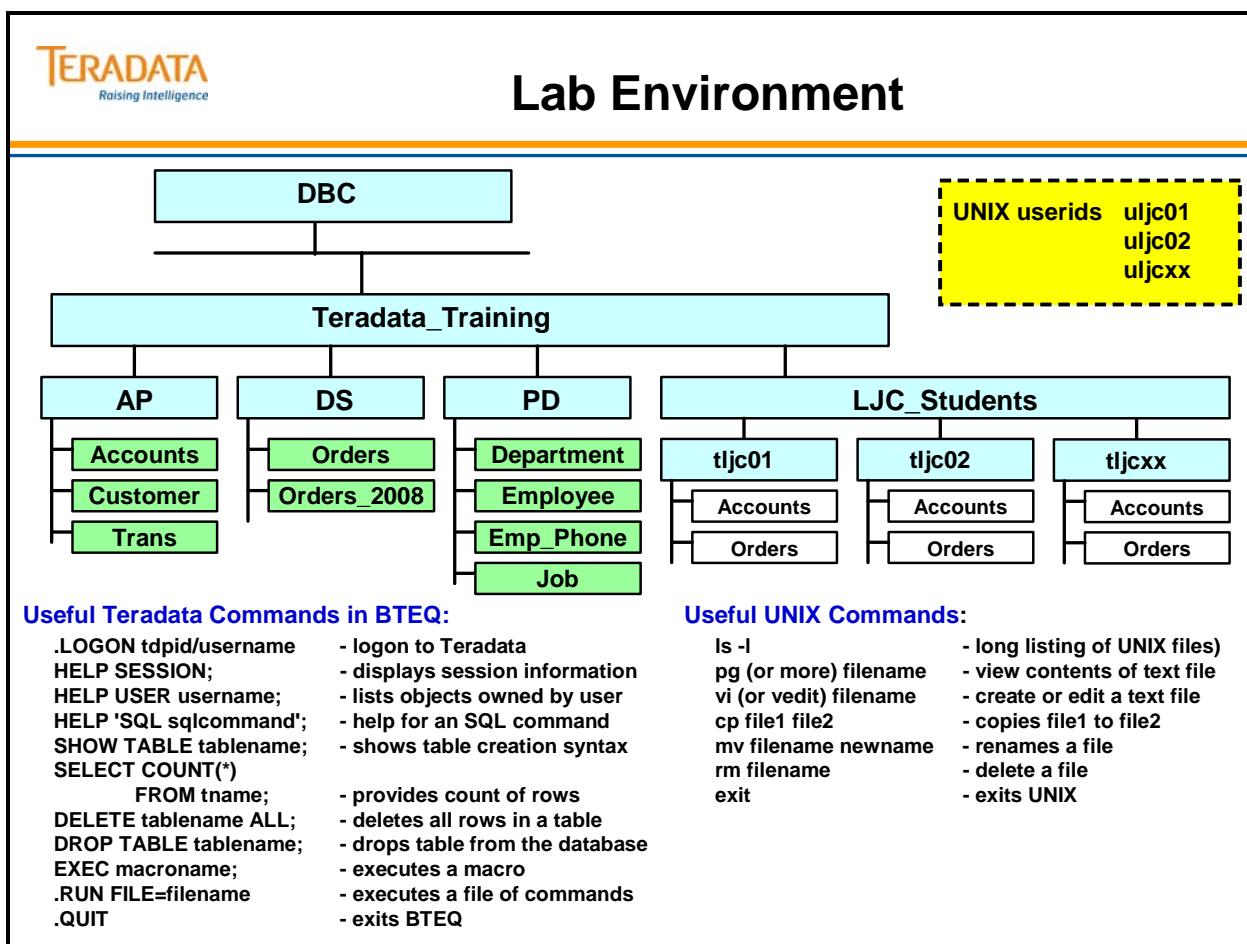
Lab Environment Notes

One way to connect from a Windows workstation to a Teradata server is to ***telnet*** to the IP address of the Teradata server. This opens a terminal window which you can use to logon to UNIX and enter UNIX commands.

Start → Run → telnet IP_address _____ ex., telnet 153.65.25.21
or telnet 153.64.115.201

Miscellaneous Notes:

- When entering your UNIX username and password, do not use the backspace key. If you enter an incorrect character, enter @ which ignores your previous input. Then enter the full username or password again.
- Once you are logged in to UNIX, you can use the backspace key to correct typing errors. You cannot use the cursor or arrow keys to correct typing errors while entering commands at a UNIX prompt.
- If you are executing BTEQ interactively, you cannot enter your password as part of the .LOGON statement. BTEQ will prompt you for your Teradata username password.



Teradata Training

Notes



AP Tables

AP.Accounts (10,000 Rows)

Account Number	Number	Street	City	State	Zip Code	Balance Forward	Balance Current
PK							
UPI							
20024010	123	Harbor Blvd.	Torrance	CA	90323	1000.00	900.00
20031023	3456	186th St.	Glendale	CA	90451	1500.00	1700.00
20049873	100	Western Av.	Las Vegas	NV	97345	400.00	400.00
20031134	10	Heather Rd.	S. Monica	CA	92345	6020.00	5312.00

AP.Customer (7,000 Rows)

Customer Number	Last Name	First Name	Social Security
PK			
UPI			
13021	Smith	George	456788765
18765	Jones	Barbara	987453498
11023	Wilson	John	495028367
1123	Omaguchi	Sandra	234904587

AP.Trans (15,000 Rows)

Trans Number	Trans Date	Account Number	Trans ID	Trans Amount
PK				
		NUPI		
4653	2002-02-11	20024020	2009	-50.00
3241	2002-02-08	20034567	DEP	160.00
1298	2002-02-08	20024005	2987	-70.00
11026	2002-02-13	20024020	DEP	20.00

Teradata Training

Notes



PD Tables

Employee (1000 Rows)

Employee Number	Dept Number	Emp_Mgr Number	Job Code	Last Name	First Name	Salary Amount
PK	FK	FK	FK			
UPI						
100001	1001	?	3000	DeBosse	Ibee	200000.00
100797	1048	100791	3017	Myers	Ruth	410000.00
100002	1001	100001	3001	Smith	Steve	110000.00
<i>Integer</i>	<i>Integer</i>	<i>Integer</i>	<i>Integer</i>	<i>Char(20)</i>	<i>Varchar(20)</i>	<i>Decimal (10,2)</i>

Job (66 Rows)

Job Code	Job Description
PK	
UPI	
3000	President
3001	Senior Mgmt
3017	Analyst L2
<i>Integer</i>	<i>Char(20)</i>

Department (60 Rows)

Dept Number	Dept Name	Dept_Mgr Number	Budget Amount
PK		FK	
UPI			
1048	Design SW	100791	1000000.00
1050	Design Services	100811	1000000.00
1028	Engineering SW	100441	3000000.00
<i>Integer</i>	<i>Char(20)</i>	<i>Integer</i>	<i>Decimal (10,2)</i>

Emp_Phone (2000 Rows)

Employee Number	Area Code	Phone Number	Extension
PK			
FK			
NUPI			
100001	937	5100001	1001
100001	937	4100001	1001
100389	858	4852815	815
<i>Integer</i>	<i>SmallInt</i>	<i>Integer</i>	<i>Integer</i>

Teradata Training

Notes



DS Tables

Orders (12,000 Rows)

o_orderid	o_custid	o_orderstatus	o_totalprice	o_ordedate	o_orderrpriority	o_clerk	o_location	o_shippriority	o_comment
PK	FK,NN								
UPI									
100001	1001	C	1,005.00	2000-01-02	10	Jack ..	5	20	In Stock
103501	1451	C	1,005.00	2002-12-01	10	Dee ...	3	20	In Stock
101400	1080	C	1,150.00	2001-02-28	10	Fred ...	10	20	In Stock

Orders_2008 (3600 Rows) - Same Layout

vi or vedit Notes

If you are not comfortable or familiar with vi (or vedit), you can create your script in a Windows Notepad or WordPad file, copy it, and paste it into a vi (or vedit) file.

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function

Switch to your terminal window where UNIX is running and ...

3. **vedit labx_1.btq** (or whatever filename you wish)
enter an **i** by itself (do not press <Enter>)
Use the mouse to choose the **Edit ➔ Paste** function
Press the ESC key (multiple times does not hurt)
Enter **:wq** (this saves the file and exits vi or vedit)

Another technique that can be used to create UNIX scripts without using vi or vedit is to do the following:

1. Enter your commands (job/script) in a Notepad file.
2. Highlight the text and use the mouse to choose the **Edit ➔ Copy** function

Switch to your terminal window where UNIX is running and ...

3. **cat > labx_1.btq** (or whatever filename you wish)
Use the mouse to choose the **Edit ➔ Paste** function
To exit the cat command, press either the DELETE key or CNTL C.

vi (or vedit) Commands

vi filename
vedit filename

Movement and other

h	move cursor to left
j	move cursor down a line
k	move cursor up a line
l	move cursor to right
space	move cursor to right
w	move a word to right
b	move a word to left
0	go to start of a line (zero)
\$	go to end of a line
G	go to end of file
nG	go to line n
^f	scroll forward 1 screen
^b	scroll back 1 screen
^l	refresh the screen
/str	search forward for str
.	repeat last command

Modification

r	replace 1 character
x	delete char under cursor
X	delete char before cursor
5x	delete 5 characters
dw	delete a word
dd	delete the line
d\$	delete the rest of the line
dG	delete rest of the file
cw	change a word
C	change rest of the line
J	join 2 lines together
yy	yank or copy a line
3yy	yank or copy 3 lines
p	put yanked line(s) after line
P	put yanked line(s) before line
u	undo last command

Input mode

i	insert before cursor
I	insert at beginning of line
a	insert after cursor
A	insert at end of line
o	open a new line below
O	open a new line above
R	Replace mode and insert after <cr>

ESC exits
input &
colon
modes

Colon mode

:w	write or save the file
:w fname	write new file name
:wq	write and quit
:q!	quit and don't save
:	
:e fname	edit another file
:r fname	read in a file
:!cmd	run UNIX command
:set smd	set showmode on

Teradata Training

Notes

Module B



Appendix B: Solutions to Lab Exercises

**This Appendix contains possible solutions
to the lab exercises in Binder #2.**

Teradata Proprietary and Confidential

Teradata Training

Notes



Lab Solutions for Lab 30-1

Lab Exercise 30-1

1. Use INSERT/SELECT to place all rows from the populated PD tables into your empty tables. Verify the number of rows in your tables.

```
INSERT INTO Employee SELECT * FROM PD.Employee;  
SELECT COUNT(*) FROM Employee; Count = 1000
```

```
INSERT INTO Department SELECT * FROM PD.Department;  
SELECT COUNT(*) FROM Department; Count = 60
```

```
INSERT INTO Job SELECT * FROM PD.Job;  
SELECT COUNT(*) FROM Job; Count = 66
```

2. EXPLAIN the following SQL statement.

```
SELECT Last_Name, First_Name, Dept_Name, Job_Desc  
FROM Employee E  
INNER JOIN Department D ON E.Dept_Number = D.Dept_Number  
INNER JOIN Job J ON E.Job_Code = J.Job_Code  
ORDER BY 3, 1, 2;
```

What is estimated time cost for this EXPLAIN? 0.09 seconds



Lab Solutions for Lab 30-1 (cont.)

Lab Exercise 30-1 (cont.)

3. Create a “non-compressed” join index which includes the following columns of Employee, Department, and Job.

```
CREATE JOIN INDEX EDJ_JI AS
SELECT      Last_Name, First_Name, D.Dept_Number, Dept_Name, J.Job_Code, Job_Desc
FROM        Employee E
INNER JOIN  Department D      ON E.Dept_Number = D.Dept_Number
INNER JOIN  Job J            ON E.job_code = J.job_code;
```

4. EXPLAIN the following SQL statement (same SQL as step #2)

What is estimated time cost for this EXPLAIN? [0.04 seconds](#)

Is the join index used? [Yes](#)

How much space does the join index require (use the DBC.TableSize view)? [185,344](#)

```
SELECT      TableName , SUM(CurrentPerm)
FROM        DBC.TableSize
WHERE       DatabaseName = USER  AND TableName = 'EDJ_JI'
GROUP BY    1
ORDER BY    1;
```

TableName	Sum(CurrentPerm)
EDJ_JI	185344



Lab Solutions for Lab 30-1 (cont.)

Lab Exercise 30-1 (cont.)

5. EXPLAIN the following SQL statement – Salary_Amount has been added as a projected column.

```
SELECT      Last_Name, First_Name, Dept_Name, Job_Desc, Salary_Amount
FROM        Employee E
INNER JOIN  Department D  ON E.Dept_Number = D.Dept_Number
INNER JOIN  Job J        ON E.Job_Code = J.Job_Code
ORDER BY    3, 1, 2;
```

What is estimated time cost for this EXPLAIN? 0.09 seconds

Is the join index used? No If not, why not? Salary_Amount is not in the Join Index.

6. Drop the Join Index.

```
DROP JOIN INDEX EDJ_JI;
```



Lab Solutions for Lab 33-1

Lab Exercise 33-1

```
cat lab33_14.btq
.LOGON u4455/tljc30,tljc30
.IMPORT DATA FILE = data33_1
.QUIET ON
.REPEAT *
USING    in_account_number (INTEGER),
          in_number (INTEGER),
          in_street (CHAR(25)),
          in_city (CHAR(20)),
          in_state (CHAR(2)),
          in_zip_code (INTEGER),
          in_balance_forward (DECIMAL(10,2)),
          in_balance_current (DECIMAL(10,2))
INSERT INTO Accounts
VALUES
  (:in_account_number , :in_number , :in_street , :in_city , :in_state ,
   :in_zip_code , :in_balance_forward, :in_balance_current);
.QUIT

bteq < lab33_14.btq
```



Lab Solutions for Lab 33-1

Lab Exercise 33-1

```
cat lab33_16.btq
.SESSIONS 8
.LOGON u4455/tljc30,tljc30
.IMPORT DATA FILE = data33_1
.QUIET ON
.REPEAT *
USING    in_account_number (INTEGER),
          in_number (INTEGER),
          in_street (CHAR(25)),
          in_city (CHAR(20)),
          in_state (CHAR(2)),
          in_zip_code (INTEGER),
          in_balance_forward (DECIMAL(10,2)),
          in_balance_current (DECIMAL(10,2))
INSERT INTO Accounts
VALUES
  (:in_account_number , :in_number , :in_street , :in_city , :in_state ,
   :in_zip_code , :in_balance_forward, :in_balance_current);
.QUIT

bteq < lab33_16.btq
```



Lab Solutions for Lab 33-1

Lab Exercise 33-1

```
cat lab33_18.btq
.SESSIONS 8
.LOGON u4455/tljc30,tljc30
.IMPORT DATA FILE = data33_1
.QUIET ON
.REPEAT * PACK 10
USING    in_account_number (INTEGER),
          in_number (INTEGER),
          in_street (CHAR(25)),
          in_city (CHAR(20)),
          in_state (CHAR(2)),
          in_zip_code (INTEGER),
          in_balance_forward (DECIMAL(10,2)),
          in_balance_current (DECIMAL(10,2))
INSERT INTO Accounts
VALUES
  (:in_account_number , :in_number , :in_street , :in_city , :in_state ,
   :in_zip_code , :in_balance_forward, :in_balance_current);
.QUIT

bteq < lab33_18.btq
```



Lab Solutions for Lab 33-2

Lab Exercise 33-2

```
cat lab33_22.btq
.LOGON u4455/tljc30,tljc30
.IMPORT DATA FILE = data33_2
.EXPORT REPORT FILE = report33_2
.WIDTH 80
.REPEAT *
    USING      in_customer_number (INTEGER)
    SELECT     customer_number, last_name (CHAR(10)), first_name (CHAR(10)), social_security
    FROM       AP.Customer
    WHERE      customer_number = :in_customer_number;
.EXPORT RESET
.QUIT
```

```
more report33_2
```

CUSTOMER_NUMBER	LAST_NAME	FIRST_NAME	SOCIAL_SECURITY
9000	Underwood	Anne	213633756
:	:	:	:
CUSTOMER_NUMBER	LAST_NAME	FIRST_NAME	SOCIAL_SECURITY
8501	Atchison	Jose	213631261



Lab Solutions for Lab 34-1

Lab Exercise 34-1

`cat lab34_12.fld`

```

LOGON u4455/tljc30,tljc30;
BEGIN LOADING Customer
  ERRORFILES cust_err1, cust_err2;
DEFINE  in_cust      (INTEGER),
        in_lname    (CHAR(30)),
        in_fname    (CHAR(20)),
        in_social   (INTEGER)
FILE = data34_1;
INSERT INTO Customer    VALUES
  (:in_cust,
   :in_lname,
   :in_fname,
   :in_social);
LOGOFF;
```

Or

```

LOGON u4455/tljc30,tljc30;
BEGIN LOADING Customer
  ERRORFILES cust_err1, cust_err2;
DEFINE FILE = data34_1;
INSERT INTO Customer.* ;
LOGOFF;
```

`fastload < lab34_12.fld`

`cat lab34_13.fld`

```

LOGON u4455/tljc30,tljc30;
BEGIN LOADING Customer
  ERRORFILES cust_err1, cust_err2;
DEFINE  in_cust      (INTEGER),
        in_lname    (CHAR(30)),
        in_fname    (CHAR(20)),
        in_social   (INTEGER)
FILE = data34_2;
INSERT INTO Customer    VALUES
  (:in_cust,
   :in_lname,
   :in_fname,
   :in_social);
END LOADING;
LOGOFF;
```

Or

```

LOGON u4455/tljc30,tljc30;
BEGIN LOADING Customer
  ERRORFILES cust_err1, cust_err2;
DEFINE FILE = data34_2;
INSERT INTO Customer.* ;
END LOADING;
LOGOFF;
```

`fastload < lab34_13.fld`



Lab Solutions for Lab 34-2

Lab Exercise 34-2

```
cat lab34_22.fld
LOGON u4455/tljc30,tljc30;
BEGIN LOADING Trans ERRORFILES trans_err1, trans_err2;
DEFINE   in_transno      (INTEGER),
         in_transdate    (CHAR(10), NULLIF='0000-00-00'),
         in_accno        (INTEGER),
         in_trans_id     (CHAR(4)),
         in_trans_amt    (DECIMAL(10,2))
FILE = data34_3;
INSERT INTO Trans
VALUES ( :in_transno,
         :in_transdate    (FORMAT 'YYYY-MM-DD'),
         :in_accno,
         :in_trans_id,
         :in_trans_amt );
END LOADING;
LOGOFF;

fastload < lab34_22.fld

From bteq:
SELECT COUNT(*) FROM Trans WHERE Trans_Date IS NULL;
COUNT(*)
150
```



Lab Solutions for 35-1

Lab Exercise 35-1

```
cat lab35_12.fxp
```

```
.LOGTABLE Restartlog3512_fxp;
.LOGON u4455/tljc30,tljc30;
.ACCEPT cnum, lname, fname, ssec FROM FILE data35_1;
.SET filename TO 'Customer';
INSERT INTO &filename
    VALUES (&cnum, '&lname', '&fname', &ssec) ;
.LOGOFF;
```

```
fexp < lab35_12.fxp
```



Lab Solutions for Lab 36-1

Lab Exercise 36-1

```
cat lab36_12.fxp
.LOGTABLE Restartlog3612_fxp ;
.LOGON u4455/tljc30,tljc30 ;
.BEGIN EXPORT;
.EXPORT OUTFILE data36_1;
SELECT      T.trans_number
            ,A.account_number
            ,A.number
            ,A.street
            ,A.city
            ,A.state
            ,A.zip_code
FROM        AP.Accounts A
INNER JOIN  AP.Trans T
    ON A.Account_number = T.Account_number;
.END EXPORT;
.LOGOFF;
```

fexp < lab36_12.fxp

(alternative join syntax)

```
cat lab36_12a.fxp
.LOGTABLE Restartlog3612a_fxp ;
.LOGON u4455/tljc30,tljc30 ;
.BEGIN EXPORT;
.EXPORT OUTFILE data36_1;
SELECT      T.trans_number
            ,A.account_number
            ,A.number
            ,A.street
            ,A.city
            ,A.state
            ,A.zip_code
FROM        AP.Accounts A
, AP.Trans T
    WHERE A.Account_number = T.Account_number ;
.END EXPORT;
.LOGOFF;
```

fexp < lab36_12a.fxp



Lab Solutions for Lab 36-2 (ACCEPT and CASE)

```
cat lab36_22a.fxp
.LOGTABLE Restartlog3622a_fxp;
.LOGON u4455/tljc30,tljc30;
.SET LoVal TO 500;
.SET HiVal TO 9499;
.ACCEPT par_city FROM FILE data36_2;

.BEGIN EXPORT;
.EXPORT OUTFILE report36_a MODE RECORD FORMAT TEXT;

SELECT Account_Number (CHAR(12)),
       City (CHAR(12)),
       Balance_Current (CHAR(12)),
       (CASE WHEN Balance_Current < &LoVal THEN 'Below MIN'
             WHEN Balance_Current > &HiVal THEN 'Above MAX'
             END)
  FROM AP.Accounts
 WHERE City = '&par_city'
   AND (Balance_Current < &LoVal OR
        Balance_Current > &HiVal)
 ORDER BY Account_Number ;
.END EXPORT;
.LOGOFF;

fexp < lab36_22a.fxp
```



Lab Solutions for Lab 36-2 (ACCEPT and UNION)

```
cat lab36_22b.fxp
.LOGTABLE Restartlog3622b_fxp;
.LOGON u4455/tljc30,tljc30;
.SET LoVal TO 500;
.SET HiVal TO 9499;
.ACCEPT par_city FROM FILE data36_2;

.BEGIN EXPORT;
.EXPORT OUTFILE report36_b MODE RECORD FORMAT TEXT;

SELECT Account_Number (CHAR(12)), City (CHAR(12)), Balance_Current (CHAR(12)),
       'Above Max'
FROM AP.Accounts
WHERE City = '&par_city'
AND Balance_Current > &HiVal

UNION

SELECT Account_Number (CHAR(12)), City (CHAR(12)), Balance_Current (CHAR(12)),
       'Below Min'
FROM AP.Accounts
WHERE City = '&par_city'
AND Balance_Current < &LoVal
ORDER BY 1;
.END EXPORT;
.LOGOFF;

fexp < lab36_22b.fxp
```



Lab Solutions for Lab 36-2 (.IMPORT)

```
cat lab36_22c.fxp
.LOGTABLE Restartlog3622c_fxp;
.LOGON u4455/tljc30,tljc30;
.SET LoVal TO 500;
.SET HiVal TO 9499;

.BEGIN EXPORT;

.LAYOUT      Record_Layout;
.FIELD       in_city * CHAR(15);

.IMPORT      INFILE data36_2c FORMAT TEXT LAYOUT Record_Layout ;

.EXPORT OUTFILE report36_c MODE RECORD FORMAT TEXT;

SELECT      Account_Number (CHAR(12)),
            City (CHAR(12)),
            Balance_Current (CHAR(12)),
            (CASE WHEN Balance_Current < &LoVal THEN 'Below MIN'
                  WHEN Balance_Current > &HiVal THEN 'Above MAX'
                  END)
FROM        AP.Accounts
WHERE       City = :in_city
AND         (Balance_Current < &LoVal OR Balance_Current > &HiVal)
ORDER BY    Account_Number ;
.END EXPORT;
.LOGOFF;

fexp < lab36_22c.fxp
```



Lab Solutions for Lab 37-1

```
cat lab37_13.mld
.LOGTABLE Restartlog3713_mld;
.LOGON u4455/tljc30,tljc30 ;
.BEGIN MLOAD TABLES Accounts, Customer, Trans ;

.Layout Record_Layout;
.FIELD File_Control      *      CHAR(1) ;
.FIELD PI_Value           *      INTEGER ;

.DML LABEL Del_Acct ;
    DELETE FROM Accounts WHERE Account_Number = :PI_Value ;

.DML LABEL Del_Cust ;
    DELETE FROM Customer WHERE Customer_Number = :PI_Value ;

.DML LABEL Del_Trans ;
    DELETE FROM Trans      WHERE Account_Number = :PI_Value ;

.IMPORT INFILE data37_1
    LAYOUT Record_Layout
        APPLY Del_Acct          WHERE File_Control = 'A'
        APPLY Del_Cust           WHERE File_Control = 'C'
        APPLY Del_Trans          WHERE File_Control = 'T' ;

.END MLOAD ;
.LOGOFF ;
```

mload < lab37_13.mld > lab37_13.out



Lab Solutions for Lab 38-1

cat lab38_13.mld

```
.LOGTABLE Restartlog3813_mld ;
.LOGON u4455/tljc30,tljc30 ;
.BEGIN MLOAD TABLES
    Accounts, Customer, Trans ;
.LAYOUT Record_Layout;
.FILLER in_code      1  CHAR(1);
.FIELD  in_accountno 2  INTEGER;
.FIELD  in_number     *  INTEGER;
.FIELD  in_street     *  CHAR(25);
.FIELD  in_city       *  CHAR(20);
.FIELD  in_state      *  CHAR(2);
.FIELD  in_zip_code   *  INTEGER;
.FIELD  in_balancefor *  DECIMAL (10,2);
.FIELD  in_balancecur *  DECIMAL (10,2);
.FIELD  in_custno     2  INTEGER;
.FIELD  in_lastname   *  CHAR(30);
.FIELD  in_firstname  *  CHAR(20);
.FIELD  in_socsec     *  INTEGER;
.FIELD  in_transno    2  INTEGER;
.FIELD  in_transdate *  CHAR(10);
.FIELD  in_accid     *  INTEGER;
.FIELD  in_transid   *  CHAR(4);
.FIELD  in_transamount *  DECIMAL(10,2);
```



(continued)

```
.DML LABEL Insert_Acct ;
INSERT INTO Accounts VALUES
(:in_accountno, :in_number, :in_street,
:in_city, :in_state, :in_zip_code, :in_balancefor,
:in_balancecur) ;

.DML LABEL Insert_Cust ;
INSERT INTO Customer VALUES
( :in_custno, :in_lastname,
:in_firstname, :in_socsec ) ;

.DML LABEL Insert_Trans ;
INSERT INTO Trans VALUES
( :in_transno, :in_transdate, :in_accid,
:in_transid, :in_transamount);

.IMPORT INFILe data38_1
    LAYOUT Record_Layout
        APPLY Insert_Acct WHERE in_code = 'A'
        APPLY Insert_Cust WHERE in_code = 'C'
        APPLY Insert_Trans WHERE in_code = 'T';

.END MLOAD ;
.LOGOFF ;
```

mload < lab38_13.mld > lab38_13.out



Lab Solutions for Lab 38-2

```

cat lab38_23.mld
.LOGTABLE Restartlog3823_mld ;
.LOGON u4455/tljc30,tljc30 ;
.BEGIN IMPORT MLOAD TABLES Accounts ;
.LAYOUT Record_Layout;
.FIELD in_accountno    1   INTEGER;
.FIELD in_number        *   INTEGER;
.FIELD in_street         *   CHAR(25);
.FIELD in_city           *   CHAR(20);
.FIELD in_state          *   CHAR(2);
.FIELD in_zip_code       *   INTEGER;
.FIELD in_balancefor    *   DECIMAL (10,2);
.FIELD in_balancecur    *   DECIMAL (10,2);

.DML LABEL Fix_Account DO INSERT FOR MISSING UPDATE ROWS ;

UPDATE Accounts      SET Balance_Current = :in_balancecur
WHERE Account_Number = :in_accountno ;

INSERT INTO Accounts VALUES (:in_accountno, :in_number, :in_street, :in_city,
                            :in_state, :in_zip_code, :in_balancefor, :in_balancecur);

.IMPORT INFILE data38_2 LAYOUT Record_Layout APPLY Fix_Account;

.END MLOAD;
.LOGOFF;

mload < lab38_23.mld > lab38_23.out

```



Lab Solutions for Lab 39-1

```
cat lab39_13.tpp
.LOGTABLE Restartlog813_tpp;
.LOGON u4455/tljc30,tljc30;
.BEGIN LOAD SESSIONS 4 PACK 40 RATE 4800;
.LAYOUT Record_Layout;
.FIELD in_accountno    1   INTEGER      KEY;
.FIELD in_number        *   INTEGER;
.FIELD in_street        *   CHAR(25);
.FIELD in_city          *   CHAR(20);
.FIELD in_state         *   CHAR(2);
.FIELD in_zip_code      *   INTEGER;
.FIELD in_balancefor    *   DECIMAL (10,2);
.FIELD in_balancecur    *   DECIMAL (10,2);

.DML LABEL Fix_Account DO INSERT FOR MISSING UPDATE ROWS
  USE (in_accountno,in_number,in_street,in_city,in_state,in_zip_code,in_balancefor,in_balancecur);
  UPDATE Accounts      SET Balance_Current = :in_balancecur
    WHERE Account_Number = :in_accountno;
  INSERT INTO Accounts VALUES (:in_accountno, :in_number, :in_street, :in_city,
                               :in_state, :in_zip_code, :in_balancefor, :in_balancecur);

.IMPORT INFILE data39_1 LAYOUT Record_Layout APPLY Fix_Account;
.END LOAD;
.LOGOFF;

tpump < lab39_13.tpp | tee lab39_13.out
```



Lab Solutions for Lab 42-1

Lab Exercise 42-1

1. Using the DBC.DBColumnInfo view, find the release and version of the system on which you are logged on:

```
SELECT      *
FROM        DBC.DBColumnInfo;

InfoKey      InfoData
RELEASE      V2R.06.02.00.16      (PDE Release #)
VERSION      06.02.00.12      (Teradata Version #)
```

2. Using the DBC.Children view, list your parents' userids:

```
SELECT      Parent
FROM        DBC.Children
WHERE       Child = USER;

Parent
DBC
SysDBA
STUDENTS
LJC_Students
```



Lab Solutions for Lab 42-1

Lab Exercise 42-1 (cont.)

3. Using the DBC.Databases view, find your:

```
SELECT      *
FROM        DBC.Databases
WHERE       DatabaseName = USER ;
```

Immediate parent's name	LJC_Students
Creator's name	LJC_Students
Default account code	\$M_9038_&S_&D&H
Perm space limit	25,000,000
Spool space limit	200,000,000
Temp space limit	100,000,000

4. Using the DBC.Users view, find your:

```
SELECT      *
FROM        DBC.Users
WHERE       UserName = USER ;
```

Default database name	?
Default collation sequence	H
Default date format	?
Create Time Stamp	2007-02-05 14:22:05
Last password modification date	2007-02-05



Lab Solutions for Lab 42-1

Lab Exercise 42-1 (cont.)

4. OPTIONAL: SHOW this view. Note the WHERE conditions. (Remember, this is a restricted view, even though it does not have an [X] suffix.)

```
REPLACE VIEW DBC.Users
AS SELECT
    dbase.DatabaseName(NAMED UserName),
    dbase.CreatorName,
    ...

```

5. Using the DBC.Tables view, find the number of tables in the DD/D (User DBC) that are:

Fallback protected Count is 105

```
SELECT      Count(*)
FROM        DBC.Tables
WHERE       DatabaseName = 'DBC'
AND         TableKind = 'T' AND ProtectionType = 'F';
```

Not Fallback protected Count is 13

```
SELECT      Count(*)
FROM        DBC.Tables
WHERE       DatabaseName = 'DBC'
AND         TableKind = 'T' AND ProtectionType = 'N';
```



Lab Solutions for Lab 42-1

Lab Exercise 42-1 (cont.)

5. Modify the query to find the number of tables OTHER THAN DD/D (not DBC tables) that are:

Fallback protected Count will vary - 1065

```
SELECT  Count(*)
FROM    DBC.Tables
WHERE   DatabaseName NE 'DBC'
AND     TableKind = 'T'
AND     ProtectionType = 'F';
```

Not Fallback protected Count will vary - 390

```
SELECT  Count(*)
FROM    DBC.Tables
WHERE   DatabaseName NE 'DBC'
AND     TableKind = 'T'
AND     ProtectionType = 'N';
```



Lab Solutions for Lab 42-1

Lab Exercise 42-1 (cont.)

6. Using the DBC.Columns view, find the number of columns in the entire system defined with *default values*:

Number of columns Count will vary - 1460

```
SELECT      'Column count:'
            ,COUNT (*)
FROM        DBC.Columns
WHERE       DefaultValue IS NOT NULL;
```

<u>'Column count:'</u>	<u>Count(*)</u>
Column count:	1460

OPTIONAL: Modify the query to find the number of TABLES or VIEWS that have columns defined with *default values*:

Number of tables Count will vary - 115

```
SELECT      'Table Count:'
            ,COUNT (DISTINCT (DatabaseName || '.' || TableName))
FROM        DBC.Columns
WHERE       DefaultValue IS NOT NULL;
```

<u>'Table Count:'</u>	<u>Count(Distinct(TableName))</u>
Table Count:	115



Lab Solutions for Lab 42-1

Lab Exercise 42-1 (cont.)

7. Using the DBC.IndexConstraints view, list the tables in your database that have a PPI and display the partitioning constraints.

Number of PPI tables Count will vary – 2
Type of PPI Constraint Check – Range N

```
SELECT      TableName,  
           ConstraintType AS "Type",  
           ConstraintText AS "Text"  
      FROM      DBC.INDEXCONSTRAINTS  
     WHERE      ConstraintType = 'Q'  
     AND       DatabaseName = USER;
```

<u>TableName</u>	<u>Type</u>	<u>Text</u>
ORDERS_PPI_M	Q	CHECK ((RANGE_N(o_orderdate BETWEEN DATE '2000-01-01' AND ...
ORDERS_PPI_MWD	Q	CHECK ((RANGE_N(o_orderdate BETWEEN DATE '2000-01-01' AND ...



Lab Solutions for Lab 42-1

Lab Exercise 42-1 (cont.)

8. Using the Indices view, find the number of tables OTHER THAN Dictionary tables that have non-unique primary indexes (NUPI):

Number of tables **Count will vary - 668**

```
SELECT      COUNT (DISTINCT(DatabaseName || '.' || TableName))
FROM        DBC.Indices
WHERE       IndexType IN ('P', 'Q')
AND         UniqueFlag = 'N'
AND         DatabaseName NE 'DBC';
```

Count(Distinct(TableName))
668

Note: The IndexType of 'P' is used for the primary index of non-partitioned tables.
The IndexType of 'Q' is used for the primary index of partitioned tables.



Lab Solutions for Lab 42-2

Lab Exercise 42-2

1. Use INSERT/SELECT to place all rows from the populated PD tables into your empty tables. Verify the number of rows in your tables.

```
INSERT INTO Employee SELECT * FROM PD.Employee;  
SELECT COUNT(*) FROM Employee; Count = 1000
```

```
INSERT INTO Department SELECT * FROM PD.Department;  
SELECT COUNT(*) FROM Department; Count = 60
```

```
INSERT INTO Job SELECT * FROM PD.Job;  
SELECT COUNT(*) FROM Job; Count = 66
```

```
INSERT INTO Emp_Phone SELECT * FROM PD.Emp_Phone  
SELECT COUNT(*) FROM Emp_Phone; Count = 2000
```

2. Use the GRANT statement to GRANT yourself the REFERENCES access rights on the tables.

```
GRANT REFERENCES ON Employee TO tljc20;  
GRANT REFERENCES ON Department TO tljc20;  
GRANT REFERENCES ON Job TO tljc20;
```



Lab Solutions for Lab 42-2

Lab Exercise 42-2 (cont.)

3. Create a References constraint between the Employee.Dept_Number column and the Department.Dept_Number column.

```
ALTER TABLE Employee ADD CONSTRAINT emp_dept_ref
  FOREIGN KEY (dept_number) REFERENCES
    Department (dept_number);
```

What is the name of the Employee RI error table? **EMPLOYEE_0**

How many rows are in this table? **1**

Which department is not represented in the department table? **1000**

4. Use the DBC.ALL_RI_Children view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? **0**

```
SELECT      Indexid  (FORMAT 'z9') AS ID
            ,IndexName
            ,ChildTable
            ,ChildKeyColumn
            ,ParentTable
            ,ParentKeyColumn
  FROM        DBC.ALL_RI_Children
 WHERE       ChildDB = USER
 ORDER BY    1;
```

<u>ID</u>	<u>IndexName</u>	<u>ChildTable</u>	<u>ChildKeyColumn</u>	<u>ParentTable</u>	<u>ParentKeyColumn</u>
0	emp_dept_ref	Employee	dept_number	Department	dept_number



Lab Solutions for Lab 42-2

Lab Exercise 42-2 (cont.)

Optional Tasks

5. Create a References constraint between the Employee.Job_code column and the Job.Job_Code column.

```
ALTER TABLE Employee ADD CONSTRAINT emp_job_ref
    FOREIGN KEY (job_code) REFERENCES Job (job_code);
```

What is the name of the Employee RI error table? **EMPLOYEE_4**

How many rows are in this table? **1**

Which job code is not represented in the job table? **3000**

6. Use the DBC.All_RI_Children view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? **4**

```
SELECT      Indexid  (FORMAT 'z9') AS ID
            ,IndexName
            ,ChildTable, ChildKeyColumn
            ,ParentTable, ParentKeyColumn
        FROM      DBC.ALL_RI_Children
        WHERE     ChildDB = USER
        ORDER BY  1 ;
```

ID	IndexName	ChildTable	ChildKeyColumn	ParentTable	ParentKeyColumn
0	emp_dept_ref	Employee	dept_number	Department	dept_number
4	emp_job_ref	Employee	job_code	Job	job_code



Lab Solutions for Lab 42-2

Lab Exercise 42-2 (cont.)

Optional Tasks (cont.)

7. Create a References constraint between the Employee.Emp_Mgr_Number column and the Employee.Employee_Number column.

```
ALTER TABLE Employee ADD CONSTRAINT emp_mgr_ref
FOREIGN KEY (emp_mgr_number) REFERENCES Employee (employee_number);
```

What is the name of the Employee RI error table? EMPLOYEE_8

How many rows are in this table? 1

Which employee does not have a manager (Emp_Mgr_Number is 0 or NULL)? 100001

```
SELECT * FROM Employee WHERE emp_mgr_number = 0;
```

8. Use the DBC.All_RI_Children view (qualify the ChildDB to your database) and verify this References constraint.

What is the IndexID of this constraint? 8

```
SELECT      Indexid  (FORMAT 'z9') AS ID ,IndexName
           ,ChildTable
           ,ChildKeyColumn
           ,ParentTable
           ,ParentKeyColumn
FROM        DBC.ALL_RI_Children
WHERE       ChildDB = USER
ORDER BY    1 ;
```

ID	IndexName	ChildTable	ChildKeyColumn	ParentTable	ParentKeyColumn
0	emp_dept_ref	Employee	dept_number	Department	dept_number
4	emp_job_ref	Employee	job_code	Job	job_code
8	emp_mgr_ref	Employee	emp_mgr_number	Employee	employee_number



Lab Solutions for Lab 44-1

Lab Exercise 44-1

1. Using the DBC.DiskSpace view, find the total disk storage capacity of the system on which you are logged on:

Total capacity 667,372,752,080

```
SELECT SUM(MaxPerm) (FORMAT 'zzz,zzz,zzz,999') FROM DBC.DiskSpace;
```

Sum(MaxPerm)

667,372,752,080

2. Using the same view, find how much of the space is currently in use:

Current space utilization 124,473,320,488

```
SELECT SUM(CurrentPerm) (FORMAT 'zzz,zzz,zzz,999') FROM DBC.DiskSpace;
```

Sum(CurrentPerm)

124,473,320,488

Write a query to show what percentage of system capacity is currently in use.

```
SELECT ( (SUM(CurrentPerm) ) / NULLIFZERO (SUM(MaxPerm)) * 100) (FORMAT 'z9.99')
       AS "% Used Perm Space"
  FROM    DBC.DiskSpace;
```

% Used Perm Space

18.65



Lab Solutions for Lab 44-1

Lab Exercise 44-1 (cont.)

2. (cont.)

OPTIONAL: Write a query to show which databases/users are currently using (current perm) the largest percentage of their max perm space limit (group by database/user).

```
SELECT      Databasename
           ,( (SUM(CurrentPerm) ) / NULLIFZERO (SUM(MaxPerm)) * 100) (FORMAT 'z9.99')
           AS "% Used Perm Space"
  FROM      DBC.DiskSpace
 GROUP BY  1
 ORDER BY  2 DESC;
```

<u>DatabaseName</u>	<u>% Used Perm Space</u>
WOW_Test	92.38
TWM_Stat_Tests_DB	87.31
AU	79.19
dbcmgr	64.82
DLM	64.55
WOW_base	49.98



Lab Solutions for Lab 44-1

Lab Exercise 44-1 (cont.)

3. Using the DBC.DataBases view, find the total number of databases and users defined in the system.

```
SELECT      COUNT(*) AS "Total Count"  
FROM        DBC.DataBases;
```

Total Count

359

Total row count (databases and users) 359 (Answers will vary)

How many users are there? 330 (Answers will vary)

```
SELECT      COUNT(*) AS "User Count"  
FROM        DBC.DataBases  
WHERE       DBKind = 'U';
```

User Count

330

How many databases are there ? 29 (Answers will vary)

```
SELECT      COUNT(*) AS "DB Count"  
FROM        DBC.DataBases  
WHERE       DBKind = 'D';
```

DB Count

29



Lab Solutions for Lab 44-1

Lab Exercise 44-1 (cont.)

3. (cont.)

Who is the creator of AP? Teradata_Training
Who is the owner of AP? Teradata_Training

```
SELECT      CreatorName, OwnerName  
FROM        DBC.DataBases  
WHERE       DatabaseName = 'AP';
```

CreatorName OwnerName
Teradata_Training Teradata_Training



Lab Solutions for Lab 44-1

Lab Exercise 44-1 (cont.)

4. Using the TableSize view, find the name and size of each table in the DBC user. List the Data Dictionary tables in DESCending order by size.

```
SELECT      TableName, SUM(CurrentPerm) (FORMAT 'zzz,zzz,zzz')
FROM        DBC.TableSize
WHERE       DatabaseName = 'DBC'
GROUP BY   1
ORDER BY   2 DESC;
```

<u>TableName</u>	<u>Sum(CurrentPerm)</u>
ResUsageSvpr	5,807,298,560
DBQLObjTbl	745,129,984
EventLog	524,814,336
DBQLogTbl	333,177,856
DBQEExplainTbl	287,761,408
ResUsageSpma	253,077,504

5. Using the DBC.AMPUsage view, find the number of AMP vprocs defined on your system.

(HINT: Use a WHERE condition to reduce the number of DD/D table rows considered.)

```
SELECT COUNT(DISTINCT (Vproc))      AS "# of AMPS"
FROM   DBC.AMPUsage
WHERE  UserName = USER;
```

<u># of AMPS</u>
20



Lab Solutions for Lab 44-1

Lab Exercise 44-1 (cont.)

6. Using the DBC.AccountInfo view, list all of your valid account codes.

```
SELECT      *
FROM        DBC.AccountInfo
WHERE       UserName = USER;
```

<u>UserName</u>	<u>AccountName</u>
tljc30	\$M_9038_&S_&D&H
tljc30	\$L_9038_&S_&D&H

7. Using the DBC.AMPUsage view, write a query to show the number of AMP CPU seconds and logical disk I/Os that have been charged to your:

```
SELECT      UserName      (CHAR(12))
           ,SUM (CPUTime)   (FORMAT 'zzzz.99')
           ,SUM (DiskIO)    (FORMAT 'zzz,zzz,zzz')
FROM        DBC.AMPUsage
WHERE       UserName = USER
GROUP BY   UserName ;
```

<u>UserName</u>	<u>Sum(CpuTime)</u>	<u>Sum(DiskIO)</u>
TLJC30	511.82	676,505



Lab Solutions for Lab 46-1

Lab Exercise 46-1

1. Using the DBC.AllRights view, find the total number of rows in the DBC.AccessRights table.

`SELECT COUNT(*) FROM DBC.AllRights;`

Count(*)
37543 Total row count (AllRights view) (Answers will vary)

Using the DBC.AllRoleRights view, find the total number of rows in the DBC.AccessRights table assigned to roles.

`SELECT COUNT(*) FROM DBC.AllRoleRights;`

Count(*)
1913 Total row count (AllRoleRights view) (Answers will vary)

2. Using the DBC.UserRights view, take a look at the databases and tables on which you currently hold rights.

`SELECT COUNT(*) FROM DBC.UserRights;`

Count(*)
101 Total number of rows returned (Answers will vary)

How do you think most of these privileges were granted?

[These are automatic rights that you received when your user and tables were created.](#)



Lab Solutions for Lab 46-1

Lab Exercise 46-1 (cont.)

2. (cont.)

Execute the following SQL command and then recheck the number of Access Rights you have.

```
CREATE TABLE Emp_Phone2 AS PD.Emp_Phone WITH NO DATA;
```

Total number of user rights returned 112 (Answers will vary)

```
SELECT COUNT(*) FROM DBC.UserRights;
```

Count(*)
112

How many new access rights were created? 11



Lab Solutions for Lab 46-1

Lab Exercise 46-1 (cont.)

3. For your Emp_Phone2 table, use the GRANT command to give the SELECT access right to the user AP.

```
GRANT SELECT ON Emp_Phone2 TO AP ;
```

Use the GRANT command to give SELECT WITH GRANT access right to the user PD.

```
GRANT SELECT ON Emp_Phone2 TO PD WITH GRANT OPTION;
```

Check the total number of user rights returned 112

```
SELECT COUNT(*) FROM DBC.UserRights;
```

Count(*)

112

Did this count change? No

If not, why not? These new rights are associated with users different than yourself.

Use the DBC.UserGrantedRights view to show any user rights that you may have explicitly granted. (Qualify this view with a WHERE clause and specify the Grantee as AP or PD.)

```
SELECT TableName, Grantee, AccessRight, GrantAuthority
FROM DBC.UserGrantedRights
WHERE Grantee IN ('AP', 'PD');
```

<u>TableName</u>	<u>Grantee</u>	<u>AccessRight</u>	<u>GrantAuthority</u>
Emp_Phone2	PD	R	Y
Emp_Phone2	AP	R	N



Lab Solutions for Lab 46-2

Lab Exercise 46-2

1. Use the DBC.RoleInfo and DBC.RoleInfoX views to display information about roles in the system.

```
SELECT      RoleName, CreatorName, CreateTimeStamp
FROM        DBC.RoleInfo
ORDER BY    1;

SELECT      RoleName, CreatorName, CreateTimeStamp
FROM        DBC.RoleInfoX
ORDER BY    1;
```

What is the primary difference between using these 2 views?

RoleInfoX only displays roles you (the user) have created.

Using the DBC.RoleInfo view, find the total number of roles defined in the system.

Using this view, how many roles are there? _____

```
SELECT COUNT(*) AS "Role Count" FROM DBC.RoleInfo;
Role Count
54  (Answers will vary.)
```

2. Grant the following access rights to the specified roles as follows:

GRANT SELECT	ON Orders	TO Role1_tljc02;
GRANT SELECT	ON Orders_2008	TO Role1_tljc02;
GRANT SELECT	ON Orders_PPI_M	TO Role2_tljc02;
GRANT INSERT, UPDATE, DELETE	ON Orders_PPI_M	TO Role3_tljc02;



Lab Solutions for Lab 46-2

Lab Exercise 46-2 (cont.)

3. Create a user profile with a profile name that is the same as your user name.

```
CREATE PROFILE tljc02 AS
  ACCOUNT          = '$M',
  DEFAULT DATABASE = tljc02,
  SPOOL            = 50e6,
  TEMPORARY        = 50e6,
  PASSWORD         = (EXPIRE = 91, MINCHAR = 6, MAXLOGONATTEMPTS = 4,
                      LOCKEDUSEREXPIRE = 1, REUSE = 365);
```

4. Use the DBC.ProfileInfo view to display information about profiles in the system.

```
SELECT      ProfileName
           ,DefaultAccount AS "Def Acct"
           ,DefaultDB
           ,SpoolSpace
           ,TempSpace
  FROM        DBC.ProfileInfo
 ORDER BY    1;
```



Lab Solutions for Lab 46-2

Lab Exercise 46-2

5. Create two new users in the system as follows:

```
CREATE USER tljc02_A AS  
    PERM = 0, PASSWORD = tljc02_A, PROFILE = tljc02, DEFAULT ROLE = Role1_tljc02;  
CREATE USER tljc02_B AS  
    PERM = 0, PASSWORD = tljc02_B, PROFILE = tljc02, DEFAULT ROLE = Role2_tljc02;
```

6. Grant "Role1" to "User_A". GRANT Role1_tljc02 to tljc02_A;
Grant "Role2" to "User_B". GRANT Role2_tljc02 to tljc02_B;
Grant "Role2" to "Role3". GRANT Role2_tljc02 TO Role3_tljc02;
Grant "Role3" to "User_B". GRANT Role3_tljc02 to tljc02_B;

7. Logon to Teradata as "User_A".

Were you prompted to enter a new password? YES

Why were you prompted to enter a new password? Because EXPIRE was not equal to 0.

8. As "User_A", execute the following SQL statements and indicate if SELECT is allowed or not.

SELECT COUNT(*) FROM Orders;	Permitted or not? <u>YES</u>
SELECT COUNT(*) FROM Orders_PPI_M;	Permitted or not? <u>NO</u>



Lab Solutions for Lab 46-2

Lab Exercise 46-2 (cont.)

9. As "User_A", use the DBC.RoleMembersX and DBC.UserRoleRights views to view the current role of the user, any nested roles, and access rights for the roles.

```
SELECT * FROM DBC.RoleMembersX;  
  
SELECT RoleName, DatabaseName, TableName, ColumnName, AccessRight  
FROM DBC.UserRoleRights  
ORDER BY 1;
```

Are there any other roles that "User_A" has available? No

Are there any nested role rights? No

How many user role rights are available to "User_A"? 2

OPTIONAL

10. Logon as "User_B" and set the password if requested.

11. As "User_B", execute the following SQL statements and indicate if SELECT is allowed or not.

SELECT COUNT(*) FROM Orders;	Permitted or not? <u>No</u>
SELECT COUNT(*) FROM Orders_PPI_M;	Permitted or not? <u>Yes</u>
DELETE Orders_PPI_M;	Permitted or not? <u>No</u>



Lab Solutions for Lab 46-2

Lab Exercise 46-2 (cont.)

12. As "User_B", use the DBC.RoleMembersX and DBC.UserRoleRights views to view the current role of the user, any nested roles, and access rights for the roles.

```
SELECT *      FROM DBC.RoleMembersX;
```

```
SELECT      RoleName, DatabaseName, TableName, ColumnName, AccessRight
  FROM      DBC.UserRoleRights
 ORDER BY    1;
```

Are there any other roles that "User_B" has available? Yes

Are there any nested role rights? No

How many user role rights are available to "User_B"? 1

13. As "User_B", use the SET ROLE command to set the current role to "Role_3".

```
SET ROLE Role3_tljc02;
```

```
SELECT COUNT(*) FROM Orders;
SELECT COUNT(*) FROM Orders_PPI_M;
DELETE Orders_PPI_M;
```

Permitted or not? No

Permitted or not? Yes

Permitted or not? Yes

14. Log off as "User_A" and "User_B". Using your initial user logon name, DROP the two users and the profile you created.

```
DROP USER tljc02_A;  DROP USER tljc02_B;  DROP PROFILE tljc02;
```



Lab Solutions for Lab 49-1

Lab Exercise 49-1

1. What system security defaults are in effect for your system? **SELECT * FROM DBC.SecurityDefaults;**

Number of days to expire password:	<u>0</u>
Minimum number of characters required:	<u>4</u>
Maximum number of characters required:	<u>30</u>
Are digits allowed?	Yes <input checked="" type="checkbox"/> No _____
Are special characters allowed?	Yes <input checked="" type="checkbox"/> No _____
Maximum failed logons permitted (0=never lock):	<u>4</u>
Minutes to elapse before unlocking:	<u>3</u>
Days to expire before password reuse:	<u>0</u>

2. Are these the security defaults that are in effect for your username? Yes or No.

SELECT * FROM DBC.ProfileInfo; (minimum password length and unlock times are different)

3. Is a Profile in effect for your username? If so, what is the name of your Profile? Student_P

SELECT Profile;

4. If a Profile is being used, which attributes in the Profile override the system security defaults?

Minimum Password Length, MaxLogonAttempts, LockedUserExpire



Lab Solutions for Lab 49-1

Lab Exercise 49-1 (cont.)

5. Using the DBC.LogOnOff view, list the “BAD” logon attempts on your system that have occurred during the last ten days. Qualify the SELECT using LIKE ‘BAD%’.

Number of Bad Logons (System) 143 (Answers will vary)

```
SELECT COUNT(*)
FROM DBC.Logonoff
WHERE EVENT LIKE 'BAD%'
AND LOGDATE > CURRENT_DATE - 10;
```

Number of Bad Logons (Your UserName) 4 (Answers will vary)

```
SELECT COUNT(*)
FROM DBC.Logonoff
WHERE EVENT LIKE 'BAD%'
AND LOGDATE > CURRENT_DATE - 10
AND UserName = USER;
```

6. Display the logon rules, if any, currently in force on your system.

```
SELECT * FROM DBC.LogonRules ORDER BY USERNAME;
```

<u>UserName</u>	<u>LogicalHostID</u>	<u>LogonStatus</u>	<u>NullPassword</u>
Crashdumps	0	G	T
Sys_Calendar	0	G	T
TDPUSER	1024	G	T
tljc28	1	R	F



Lab Solutions for Lab 49-2

Lab Exercise 49-2

Tasks

1. Using the DBC.AccLogRules view, list the access log rules that are in effect for your username.

SELECT * from DBC.AccLogRules WHERE Username=USER;

Which codes are being logged and what type of logging is being captured?

<u>Code</u>	<u>Type of Logging</u>	<u>SQL Function Being Logged</u>
Ex. <u>CDB</u>	<u>B +</u>	<u>Create Database</u>
<u>CTB</u>	<u>B +</u>	<u>Create Table</u>
<u>CUS</u>	<u>B +</u>	<u>Create User</u>
<u>DDB</u>	<u>B +</u>	<u>Drop Database</u>
<u>DTB</u>	<u>B +</u>	<u>Drop Table</u>
<u>DUS</u>	<u>B +</u>	<u>Drop User</u>
<u>EXE</u>	<u>E</u>	<u>Execute</u>
<u>CPR</u>	<u>B +</u>	<u>Create Profile</u>
<u>DPR</u>	<u>B +</u>	<u>Drop Profile</u>

2. How many different access logging rules are there for all users? Count = 72 (answers will vary)

SELECT COUNT(*) from DBC.AccLogRules;



Lab Solutions for Lab 49-2

Lab Exercise 49-2 (cont.)

Tasks

3. Execute the following statement.

```
CREATE DATABASE Test FROM DBC AS PERM=0;
```

(this command should fail – access right violation)

4. Using the DBC.AccessLog view, list the access log entries for the last two weeks for your username.

How many entries are in this log have been granted? [37 \(answers will vary\)](#)

Note: The AccLogResult column (12.0) is named "Result" in previous releases.

```
SELECT COUNT(*) from DBC.AccessLog where "Result" = 'G' and Username=USER  
AND Logdate > Current_Date - 14;
```

How many entries are in this log have been denied? [1 \(at least one; answers will vary\)](#)

```
SELECT COUNT(*) from DBC.AccessLog where "Result" = 'D' and Username=USER  
AND Logdate > Current_Date - 14;
```

What is the difference between the Create Table and the Execute command log entries?

[SQL text is not captured for Execute commands.](#)



Lab Solutions for Lab 49-2

Lab Exercise 49-2 (cont.)

Tasks

5. Using the DBC.DBQLRules view, list the attributes of query log rule that is in effect for your username.

SELECT * from DBC.DBQLRules WHERE Username=USER;

Explain Text Logged	<u>F</u>
Objects Logged	<u>F</u>
Full SQL Logged	<u>F</u>
Execution Steps Logged	<u>F</u>
Summary	<u>F</u> If Summary, times _____
Threshold	<u>T</u> If threshold, time = <u>50 (1/2 CPU second)</u>
SQL Text Size	<u>200</u>
Threshold/Summary Type	<u>1 (1 represents CPU time)</u>

6. Using the DBC.Qrylog view, list logged queries for your username and determine how many queries are logged for your username.

SELECT COUNT (*) FROM DBC.Qrylog WHERE Username = USER;

Count = 17 (answers will vary)

Using this view, how many queries are logged for all of the users with usernames like 'TLJC%'?

SELECT COUNT (*) FROM DBC.Qrylog WHERE Username LIKE 'TLJC%';

Count = 212 (answers will vary)

Module C



Appendix C: Data Dictionary Views

**This Appendix contains the
Data Dictionary Views
for
Teradata 12.0.**

Teradata Proprietary and Confidential

Teradata Training

Data Dictionary views are part of the Teradata Database Data Dictionary and reside in the space owned by the system user DBC. They provide information about users, their access rights, grants, and logons.

View definitions are stored in the table DBC.TVM. View column information is stored in DBC.TVFields.

The following are the view forms:

- Without the X (for example, DBC.AccountInfo and DBC.AccountInfoV), they display global information.
- With the X (for example, DBC.AccountInfoX and AccountInfoVX), they display information associated with the requesting user only.
- With the V (for example, AccessLogV), they display information associated with the Unicode version, where object name columns have a data type of VARCHAR(128).
- Without the V (for example, DBC.AccountInfo or DBC.AccountInfoX), they display information associated with the Compatibility version, where object name columns have a data type of CHAR(30).



Data Dictionary Views – Teradata 12.0

DBC.AccessLog[V]

User Type	Columns Selected
Security Administrator	LogDate LogTime LogonDate LogonTime LogicalHostId IFPNo SessionNo UserName AccountName OwnerName AccessType Frequency EventCount AccLogResult Result DatabaseName TVMName ColumnName StatementType StatementText QueryBand



Data Dictionary Views – Teradata 12.0

DBC.AccLogRules[V]

User Type	Columns Selected
Security Administrator	UserName DatabaseName TVMName AcrAlterFunction AcrCheckpoint AcrCreateDatabase AcrCreateFunction AcrCreateMacro AcrCreateTable AcrCreateUser AcrcreateView AcrCreateProcedure AcrCreateExtProcedure AcrDelete AcrDropDatabase AcrDropFunction AcrDropMacro AcrDropProcedure AcrDropTable AcrDropUser AcrDropView AcrDump AcrExecute AcrExecuteFunction

DBC.AccLogRules[V] (cont.)

User Type	Columns Selected
	AcrExecuteProcedure AcrGrant AcrIndex AcrInsert AcrReference AcrRestore AcrSelect AcrUpdate AcrCreateTrigger AcrDropTrigger AcrCreateRole AcrDropRole AcrCreateProfile AcrDropProfile AcrAlterProcedure AcrRepControl AcrAlterExtProcedure AcrUDTUsage AcrUDTType AcrUDTMethod AcrCreAuthorization AcrDropAuthorization CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.AccountInfo[V][X]

User Type	Columns Selected
Supervisory [X] End User Administrator	UserName UserOrProfile AccountName

DBC.AllRights[V][X]

User Type	Columns Selected
Administrator	UserName DatabaseName TableName ColumnName AccessRight GrantAuthority GrantorName AllnessFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.AllRoleRights[V][X]

User Type	Columns Selected
Security Administrator Supervisory Administrator	UserName DatabaseName TableName ColumnName AccessRight GrantAuthority GrantorName CreateTimeStamp

TERADATA
Raising Intelligence

Data Dictionary Views – Teradata 12.0

DBC>AllSpace[V][X]

User Type	Columns Selected
Administrator [X] End User Supervisory	Vproc DatabaseName AccountName TableName MaxPerm MaxSpool MaxTemp CurrentPerm CurrentSpool CurrentTemp PeakPerm PeakSpool PeakTemp



Data Dictionary Views – Teradata 12.0

DBC.AllTempTables[V][X]

User Type	Columns Selected
Administrator [X] End User	HostNo SessionNo UserName B_DatabaseName B_TableName E_TableId

TERADATA
Raising Intelligence

Data Dictionary Views – Teradata 12.0

User Type	Columns Selected
All users	IndexID IndexName ChildDB ChildTable ChildKeyColumn ParentDB ParentTable ParentKeyColumn InconsistencyFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.All_RI_Parents[V][X]

User Type	Columns Selected
All users	IndexID IndexName ParentDB ParentTable ParentKeyColumn ChildDB ChildTable ChildKeyColumn InconsistencyFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.AMPUsage[V][X]

User Type	Columns Selected
Administrator	AccountName UserName CPUTime DiskIO CPUTimeNorm Vproc VprocType Model



Data Dictionary Views – Teradata 12.0

DBC.ArchiveLoggingObjsV[X]

User Type	Columns Selected
End User	DatabaseName TVMName LogLevel CreatorName CreateTimeStamp

DBC.Association[V][X]

User Type	Columns Selected
End User	DatabaseName TableName EventNum Original_DataBaseName Original_TableName Original_TableKind Original_Version Original_ProtectionType Original_JournalFlag Original_CreatorName Original_CommentString



Data Dictionary Views – Teradata 12.0

DBC.Authorizations[V][X]

User Type	Columns Selected
Operational Control	DatabaseName AuthorizationName AuthorizationID TableKind Version AuthorizationType AuthorizationSubType OSDomainName OSUserName

DBC.CharSets[V]

User Type	Columns Selected
End User	CharSetName



Data Dictionary Views – Teradata 12.0

DBC.CharTranslations[V]

User Type	Columns Selected
End User	CharSetName CharSetId InstallFlag E2I E2IUp I2E I2EUp

DBC.Children[V][X]

User Type	Columns Selected
Administrator	Child Parent



Data Dictionary Views – Teradata 12.0

DBC.Collations[V]

User Type	Columns Selected
End User	CollName CollInstall CollEqvClass CollOrderCS CollOrderUC



Data Dictionary Views – Teradata 12.0

DBC.Columns[V][X]

User Type	Columns Selected
[X] End User Administrator	DatabaseName TableName ColumnName ColumnFormat ColumnTitle ColumnType ColumnUDTName (V2R6) ColumnLength DefaultValue Nullable CommentString DecimalTotalDigits DecimalFractionalDigits ColumnId UpperCaseFlag Compressible CompressValue ColumnConstraint ConstraintCount

DBC.Columns[V][X] (cont.)

User Type	Columns Selected
[X] End User Administrator	CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp CharType IdColType CompressValueList AccessCount LastAccessTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.ColumnStats[V]

User Type	Columns Selected
Tools	DatabaseName1 FieldID FieldFormat FieldName FieldStatistics FieldType Implied Point MaxLength TotalDigits TVMName1



Data Dictionary Views – Teradata 12.0

DBC.CostProfiles_v

User Type	Columns Selected
All Users	ProfileTypeName ProfileName ProfileCat ProfileDesc

DBC.CostProfileTypes_v

User Type	Columns Selected
All Users	ProfileTypeName ProfileTypeDesc



Data Dictionary Views – Teradata 12.0

DBC.CostProfileValues_v

User Type	Columns Selected
All Users	ProfileName ProfileId ConstName ConstId ConstCat ConstVal ConstDesc

DBC.CSPSessionInfo[V]

User Type	Columns Selected
Operations and Recovery Control	SessionNo HostNo StartMBox LogonSource



Data Dictionary Views – Teradata 12.0

DBC.Databases[V][X]

User Type	Columns Selected
[X] End User Administrator	DatabaseName CreatorName OwnerName AccountName ProtectionType JournalFlag PermSpace SpoolSpace TempSpace CommentString CreateTimeStamp LastAlterName LastAlterTimeStamp DBKind AccessCount LastAccessTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.Databases2[V][X]

User Type	Columns Selected
All users	DatabaseName DatabaseId UnResolvedRICount

DBC.Database_Default_Journals[V][X]

User Type	Columns Selected
[X] End User Administrator	DatabaseName Journal_DB JournalName

DBC.DBColumnInfo[V]

User Type	Columns Selected
All Users	InfoKey InfoData



Data Dictionary Views – Teradata 12.0

DBC.DBQLRules[V]

User Type	Columns Selected
Administrator Supervisory	UserName AccountString ExplainFlag ObjFlag SqlFlag StepFlag SummaryFlag ThresholdFlag TextSizeLimit SummaryVal1 SummaryVal2 SummaryVal3 ThreshValue

DBC.DeleteAccessLog[V][X]

User Type	Columns Selected
Security Administrator	LogDate LogTime



Data Dictionary Views – Teradata 12.0

DBC.DeleteOldInDoubt[V]

User Type	Columns Selected
Administrator	LogicalHostId CoordTaskId LogonUserName CommitOrRollback CompletionDate UserLogonTime SessionNumber RunUnitId ResolvingUserLogonName UserLogonDate CompletionTime Options



Data Dictionary Views – Teradata 12.0

DBC.DiskSpace[V][X]

User Type	Columns Selected
Administrator [X] End User Supervisory	Vproc DatabaseName AccountName MaxPerm MaxSpool MaxTemp CurrentPerm CurrentSpool CurrentTemp PeakPerm PeakSpool PeakTemp MaxProfileSpool MaxProfileTemp

The screenshot shows a Teradata Data Dictionary Views interface. At the top left is the Teradata logo with the tagline "Raising Intelligence". The main title is "Data Dictionary Views – Teradata 12.0". Below the title, the view name "DBC.ErrorTbIsV[X]" is displayed. A table lists the columns selected for the Administrator user type, which include LogicalHostId, ErrTblDbName, ErrTblName, BaseTblDbName, BaseTblName, CreatorName, and CreateTimeStamp.

User Type	Columns Selected
Administrator	LogicalHostId ErrTblDbName ErrTblName BaseTblDbName BaseTblName CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.Events[V][X]

User Type	Columns Selected
Operations and Recovery Control	CreateDate CreateTime EventNum EventType UserName DatabaseName ObjectType AllAMPsFlag RestartSeqNum OperationInProcess TableName CheckpointName LinkingEventNum DataSetName LockMode JournalUsed JournalSaved IndexPresent DupeDumpSet



Data Dictionary Views – Teradata 12.0

DBC.Events_Configuration[V][X]

User Type	Columns Selected
Operations and Recovery Control	Vproc CreateDate CreateTime EventNum EventType UserName LogProcessor PhyProcessor ProcessorState RestartSeqNum



Data Dictionary Views – Teradata 12.0

DBC.Events_Media[V][X]

User Type	Columns Selected
Operations and Recovery Control	CreateDate CreateTime EventNum EventType UserName DataSetName VolSerialId VolSequenceNum DupeDumpSet



Data Dictionary Views – Teradata 12.0

DBC.ExternalSPs[V][X]

User Type	Columns Selected
Teradata Database Administrator	DatabaseName ExternalProcedureName ExternalProcedureID NumParameters ExternalName SrcFileLanguage NoSQLDataAccess ParameterStyle ExecProtectionMode ExtFileReference CharacterType Platform RoutineKind ParameterUDTIDs AuthIDUsed AppCategory



Data Dictionary Views – Teradata 12.0

DBC.Functions[V][X]

User Type	Columns Selected
Administrator	DatabaseName FunctionName SpecificName FunctionId NumParameters ParameterDataTypes FunctionType ExternalName SrcFileLanguage NoSQLDataAccess ParameterStyle DeterministicOpt NullCall PrepareCount ExecProtectionMode ExtFileReference CharacterType PlatformInterimFldSize RoutineKind ParameterUDTIds MaxOutParameters



Data Dictionary Views – Teradata 12.0

DBC.HostsInfo[V]

User Type	Columns Selected
End User Administrator	LogicalHostId HostName DefaultCharSet

DBC. IndexConstraints[V]

User Type	Columns Selected
End User Administrator	DatabaseName TableName IndexName IndexNumber ConstraintType ConstraintText ConstraintCollation CollationName CreatorName CreateTimestamp



Data Dictionary Views – Teradata 12.0

DBC.IndexStats[V]

User Type	Columns Selected
End User Administrator	DatabaseName1 FieldFormat FieldName FieldPosition FieldType ImpliedPoint IndexNumber IndexType IndexStatistics MaxLength Name TotalDigits TVMName1 UniqueFlag



Data Dictionary Views – Teradata 12.0

DBC.Indices[V][X]

User Type	Columns Selected
[X] End User Administrator	DatabaseName TableName IndexNumber IndexType UniqueFlag IndexName ColumnName ColumnPosition CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp IndexMode IndexMode AccessCount LastAccessTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.InDoubtLog[V]

User Type	Columns Selected
Administrator	LogicalHostId CoordTaskId LogonUserName UserLogonDate CompletionDate CommitOrRollBack SessionNumber RunUnitId ResolvingUserLogonName UserLogonTime CompletionTime Options

DBC.Journals[V][X]

User Type	Columns Selected
[X] End User Administrator	Tables_DB TableName Journals_DB JournalName



Data Dictionary Views – Teradata 12.0

DBC.LogOnOff[V][X]

User Type	Columns Selected
Security Administrator	LogDate
Administrator	LogTime
Supervisory	UserName AccountName Event LogicalHostId IFPNo SessionNo LogonDate LogonTime LogonSource



Data Dictionary Views – Teradata 12.0

DBC.LogonRules[V]

User Type	Columns Selected
Security Administrator	UserName LogicalHostId LogonStatus NullPassword CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.MultiColumnStats[V]

User Type	Columns Selected
End User Administrator	DatabaseName TableName StatisticsId ColumnPosition ColumnName ColumnType ColumnLength ColumnFormat DecimalTotalDigits DecimalFractionDigits ColumnStatistics



Data Dictionary Views – Teradata 12.0

DBC.ProfileInfo[V][X]

User Type	Columns Selected
[X] End User Supervisory Security Administrator Administrator	ProfileName DefaultAccount DefaultDB SpoolSpace TempSpace ExpirePassword PasswordMinChar PasswordMaxChar PasswordDigits PasswordSpecChar PasswordRestrictWords MaxLogonAttempts LockedUserExpire PasswordReuse CommentString CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.QryLog[V]

User Type	Columns Selected
Administrator	ProcID CollectTimeStamp QueryID UserID UserName DefaultDatabase AcctString ExpandAccString SessionID LogicalHostID RequestNum InternalRequestNum LogonDateTime AccTimeString AccStringHour AccStringDate AppID ClientID ClientAddress QueryBand ProfileID StartTime FirstStepTime

DBC.QryLog[V] (cont.)

User Type	Columns Selected
Administrator	LastRespTime ElapsedTime NumSteps NumStepswPar MaxStepsInPar NumResultRows TotalIOcount TotalCPUTime ErrorCode ErrorText WarningOnly AbortFlag CacheFlag QueryText NumOfActiveAMPs HotAmp1CPU HotCPUAmpNumber LowAmp1CPU HotAmp1IO HotIOAmpNumber LowAmp1IO SpoolUsage



Data Dictionary Views – Teradata 12.0

DBC.QryLogEventHis[V]

User Type	Columns Selected
Administrator	ProcID CollectTimeStamp EntryTS EntryKind EntryID EntryName EventValue Activity ActivityId ActivityName ConfigId Spare1 Spare2



Data Dictionary Views – Teradata 12.0

DBC.QryLogEvents[V]

User Type	Columns Selected
Administrator	ProcID CollectTimeStamp SessionID LogicalHostID WDID OpEnvID SysConID EventTime EventCode EventSubCode EventInfo



Data Dictionary Views – Teradata 12.0

DBC.QryLogExceptions[V]

User Type	Columns Selected
Administrator	ProID CollectTimeStamp Query ID UserName SessionID RequestNum LogicalHostID AcctString WDID OpEnvID SysConID ClassificationTime ExceptionTime ExceptionValue ExceptionAction NewWDID ExceptionCode ExceptionSubCode ErrorText ExtraInfo RuleID WarningOnly



Data Dictionary Views – Teradata 12.0

DBC.QryLogExplain[V]

User Type	Columns Selected
Administrator	ProcID CollectTimeStamp Query ID ExpRowNo ExplainText



Data Dictionary Views – Teradata 12.0

DBC.QryLogObjects[V]

User Type	Columns Selected
Administrator	ProID CollectTimeStamp QueryID ObjectDatabaseName ObjectTableName ObjectColumnName ObjectID ObjectNum ObjectType FreqofUse TypeofUse (V2R6)

DBC.QryLogSQL[V]

User Type	Columns Selected
Administrator	ProID CollectTimeStamp Query ID SqlRowNo SqlTextInfo



Data Dictionary Views – Teradata 12.0

DBC.QryLogSteps[V]

User Type	Columns Selected
Administrator	ProcID CollectTimestamp QueryID StepLev1Num StepLev2Num StepName StepStartTime StepStopTime ElapsedTime EstProcTime EstCPUCost CPUtime IOcount EstRowCount RowCount RowCount2 NumOfActiveAMPs MaxAmpCPUTime MaxCPUAmpNumber MinAmpCPUTime MaxAmpIO MaxIOAmpNumber MinAmpIO

DBC.QryLogSteps[V] (cont.)

User Type	Columns Selected
Administrator	LastRespTime SpoolUsage MaxAMPSpool MaxSpoolAmpNumber MinAMPSpool StepWD LSN UtilityTableId RowsWComprColumns EstIOCost EstNetCost EstHRCost CPUTimeNorm MaxAmpCPUTimeNorm MaxCPUAmpNumberNorm MinAmpCPUTimeNorm



Data Dictionary Views – Teradata 12.0

DBC.QryLogSummary[V]

User Type	Columns Selected
Administrator	ProID CollectTimeStamp UserID AcctString LogicalHostID AppID ClientID ClientAddr ProfileID SessionID QueryCount ValueType QuerySeconds AverageTime TotalIOCount AverageIO AMPCPUTime AverageAmpCPU

DBC.QryLogSummary[V] (cont.)

User Type	Columns Selected
Administrator	ParserCPUTime AverageParserCPU AMPCPUTimeNorm AverageAmpCPUNorm ParserCPUTimeNorm AverageParserCPUNorm LowHist HighHist

Data Dictionary Views – Teradata 12.0	
DBC.QryLogTDWM[V]	
User Type	Columns Selected
Administrator	ProcID CollectTimeStamp QueryID UserID UserName DefaultDatabase AcctString LastStateChange DelayTime WDID OpEnvID SysConID LSN NoClassification WDOVERRIDE SLGMet ExceptionValue FinalWDID
DBC.QryLogTDWM[V] (cont.)	
User Type	Columns Selected
Administrator	TDWMEstMaxRows TDWMEstLastRows TDWMEstTotalTime TDWMAllAmpFlag TDWMConfLevelUsed



Data Dictionary Views – Teradata 12.0

DBC.QryLogTDWMSum[V]

User Type	Columns Selected
Security Administrator	ProcID CollectTimeStamp WDID OpEnvID SysConID StartColTime Arrivals ActiveCount Completions MinRespTime MaxRespTime AvgRespTime MinCPUTime MaxCPUTime AvgCPUTime DelayedCount AvgDelayTime ExceptionCount AbortCount ErrorCount OtherCount MetSLGCount



Data Dictionary Views – Teradata 12.0

DBC.RCC_Configuration[V][X]

User Type	Columns Selected
Operations and Recovery Control	EventNum LogProcessor PhyProcessor ProcessorState RestartSeqNum Vproc

DBC.RCC_Media[V][X]

User Type	Columns Selected
Operations and Recovery Control	EventNum VolSerialId VolSequenceNum DupeDumpSet



Data Dictionary Views – Teradata 12.0

DBC.RepTables[V][X]

User Type	Columns Selected
All users	GroupName TableName

DBC.RestrictedWords[V]

User Type	Columns Selected
All users	RestrictedWord



Data Dictionary Views – Teradata 12.0

DBC.RI_Child_Tables[V][X]

User Type	Columns Selected
All users	IndexID IndexName ChildDbID ChildTID ChildKeyFID ParentDbID ParentTID ParentKeyFID InconsistencyFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.RI_Distinct_Children[V][X]

User Type	Columns Selected
All users	IndexID IndexName ChildDB ChildTable ParentDB ParentTable InconsistencyFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.RI_Distinct_Parents[V][X]

User Type	Columns Selected
All users	IndexID IndexName ParentDB ParentTable ChildDB ChildTable InconsistencyFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.RI_Parent_Tables[V][X]

User Type	Columns Selected
All users	IndexID IndexName ParentDbID ParentTID ParentKeyFID ChildDbID ChildTID ChildKeyFID InconsistencyFlag CreatorName CreateTimeStamp

TERADATA
Raising Intelligence

Data Dictionary Views – Teradata 12.0

DBC.RoleInfo[V][X]	
User Type	Columns Selected
Security Administrator [X] End User	RoleName CreatorName CommentString CreateTimeStamp ExtRole (V2R6)

DBC.RoleMembers[V][X]	
User Type	Columns Selected
Security Administrator [X] End User	RoleName Grantee GranteeKind Grantor WhenGranted DefaultRole WithAdmin



Data Dictionary Views – Teradata 12.0

DBC.SecurityDefaults[V]

User Type	Columns Selected
Security Administrator	ExpirePassword PasswordMinChar PasswordMaxChar PasswordDigits PasswordSpecChar PasswordRestrictWords MaxLogonAttempts LockedUserExpire PasswordReuse

DBC.SecurityLog[V][X]

User Type	Columns Selected
Security Administrator	LogDate LogTime LogType UserName AccountName DatabaseName TableName Text

TERADATA <i>Raising Intelligence</i>		Data Dictionary Views – Teradata 12.0
DBC.SessionInfo[V][X]		
User Type	Columns Selected	
Administrator	UserName AccountName SessionNo DefaultDataBase IFPNo Partition LogicalHostId HostNo CurrentCollation LogonDate LogonTime LogonSequenceNo LogonSource ExpiredPassword TwoPCMode Transaction_Mode CurrentRole ProfileName LogonAcct LDAP AuditTrailID CurlolationLevel QueryBand	
Security Administrator		
Supervisory		
[X] End User		



Data Dictionary Views – Teradata 12.0

DBC.ShowColChecks[V][X]

User Type	Columns Selected
End User Administrator	DatabaseName TableName ColumnName ColCheck CreatorName CreateTimeStamp

DBC.ShowTblChecks[V][X]

User Type	Columns Selected
End User Administrator	DatabaseName TableName CheckName TblCheck CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.Software_Event_Log[V]

User Type	Columns Selected
Operations and Recovery Control	TheDate TheTime Event_Tag Category Severity PMA Vproc Partition Task TheFunction Function SW_Version Line Text



Data Dictionary Views – Teradata 12.0

DBC.Table_LevelConstraints[V][X]

User Type	Columns Selected
End user Administrator	DatabaseName TableName ConstraintName ConstraintText CreatorName CreateTimeStamp

TERADATA <i>Raising Intelligence</i>	
Data Dictionary Views – Teradata 12.0	
DBC.Tables[V][X]	
User Type	Columns Selected
[X] End User Administrator	DatabaseName TableName Version TableKind ProtectionType JournalFlag CreatorName RequestText CommentString ParentCount ChildCount NamedTblCheckCount UnnamedTblCheckExist PrimaryKeyIndexId RepStatus CreateTimeStamp LastAlterName LastAlterTimeStamp RequestTxtOverFlow AccessCount LastAccessTimeStamp UtilVersion QueueFlag CommitOpt TransLog



Data Dictionary Views – Teradata 12.0

DBC.Tables2[V][X]

User Type	Columns Selected
Administrator Supervisory	TVMName TVMId DatabaseId ParentCount ChildCount

DBC.TableSize[V][X]

User Type	Columns Selected
Administrator [X] End User	Vproc DatabaseName AccountName TableName CurrentPerm PeakPerm



Data Dictionary Views – Teradata 12.0

DBC.TableText[V][X]

User Type	Columns Selected
Administrator <input checked="" type="checkbox"/> End User	DatabaseName TableName TableKind RequestText LineNo



Data Dictionary Views – Teradata 12.0

DBC.Triggers[V][X]

User Type	Columns Selected
Administrator	DatabaseName SubjectTableDataBaseName TableName TriggerName EnabledFlag ActionTime Event Kind OrderNumber TriggerComment RequestText CreatorName CreateTimeStamp LastAlterName LastAlterTimeStamp AccessCount LastAccessTimeStamp CreateTxtOverflow



Data Dictionary Views – Teradata 12.0

DBC.User_Default_Journals[V][X]

User Type	Columns Selected
End User	UserName Journal_DB JournalName

DBC.UserGrantedRights[V]

User Type	Columns Selected
End User	DatabaseName TableName ColumnName Grantee GrantAuthority AccessRight AllnessFlag CreatorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.UserRights[V]

User Type	Columns Selected
End User	DatabaseName TableName ColumnName AccessRight GrantAuthority GrantorName CreatorName CreateTimeStamp

DBC.UserRoleRights[V]

User Type	Columns Selected
End User	RoleName DatabaseName TableName ColumnName AccessRight GrantorName CreateTimeStamp



Data Dictionary Views – Teradata 12.0

DBC.Users[V]

User Type	Columns Selected
Supervisory	UserName
End User	CreatorName
Administrator	PasswordLastModDate PasswordLastModTIme OwnerName PermSpace SpoolSpace TempSpace ProtectionType JournalFlag StartupScript DefaultAccount DefaultDataBase CommentString DefaultCollation PasswordChgDate LockedDate LockedTime LockedCount

DBC.Users[V] (cont.)

User Type	Columns Selected
Supervisory End User Administrator	TimeZoneHour TimeZoneMinute DefaultDateFormat CreateTimeStamp LastAlterTime LastAlterTimeStamp DefaultCharType RoleName ProfileName AccessCount LastAccessTimeStamp

Teradata Training

Notes

Module D



Appendix D: Answers to Review Questions

This Appendix contains answers to the review questions at the ends of the modules for Binder #2.

Teradata Proprietary and Confidential

Teradata Training

Notes



Module 30: Review Question Answers

Check the box if the attribute applies to the index.

	Compressed Join Index Syntax	Non- Compressed Join Index Syntax	Aggregate Join Index	Sparse Join Index	Hash Index
May be created on a single table	✓	✓	✓	✓	✓
May be created on multiple tables	✓	✓	✓	✓	
Requires the use of SUM or COUNT functions			✓		
Requires a WHERE condition to limit rows stored in the index.				✓	
Automatically updated as base table rows are inserted or updated	✓	✓	✓	✓	✓
Automatically includes the base table PI value as part of the index					✓



Module 31: Review Question Answers

1. Which BTEQ setting controls Teradata vs. ANSI mode? [SET SESSION TRANSACTION](#)
2. Which commands will not work in ANSI mode? [BT, ET](#)
3. [True](#) or False. The SQL Flagger is just a warning device and doesn't affect command execution.
4. True or [False](#). Failure of an individual request in COMMIT mode causes the entire transaction to be rolled back.
5. [True](#) or False. Logging off during an explicit transaction without either a COMMIT or ET will always result in a ROLLBACK.
6. True or [False](#). HELP SESSION will show the session mode and the status of the SQL Flagger.
7. Where does a Volatile Temporary table get its space from? [Spool](#)
8. Where does a Global Temporary table get its space from? [Temporary](#)
9. A trigger executes (fires) when either an [INSERT](#), [UPDATE](#), or [DELETE](#) statement modifies a specified column or columns in a table.



Module 32: When Do Multiple Sessions Make Sense?

Multiple sessions improve performance ONLY for SQL requests that impact fewer than ALL AMPs.

TRANS_HISTORY

Trans_Number	Trans_Date	Account_Number	Trans_ID	Amount
PK		FK,NN		
USI		NUPI		
NUSI				

Which of the following batch requests would benefit from multiple sessions?

1. **INSERT INTO Trans_History
VALUES (:T_Nbr, DATE, :Acct_Nbr, :T_ID, :Amt);**
2. **SELECT * FROM Trans_History
WHERE Trans_Number=:Trans_Number;**
3. **DELETE FROM Trans_History
WHERE Trans_Date < DATE - 120;**
4. **DELETE FROM Trans_History
WHERE Account_Number= :Account_Number;**

Trans Type	Table or Row Lock	Multiple Sessions Useful or Not?
NUPI	Row Hash	Yes
NUSI	Full Table	No
FTS	Full Table	No
NUPI	Row Hash	Yes



Module 32: Review Question Answers

Answer True or False.

1. True or False. With MultiLoad, you can import and export data.
2. True or False. In Teradata mode, a BTEQ DELETE ALL function does not use the Transient Journal to store before-images of deleted rows.
3. True or False. An INSERT/SELECT of 1,000,000 rows into an empty table is only slightly faster than an INSERT/SELECT of 1,000,000 rows into a table with 1 row.

Match the Teradata Parallel Transporter operator with the corresponding Teradata utility.

- | | |
|--------------------|---------------|
| 1. <u>A</u> UPDATE | A. MultiLoad |
| 2. <u>D</u> STREAM | B. FastLoad |
| 3. <u>B</u> LOAD | C. FastExport |
| 4. <u>C</u> EXPORT | D. TPump |



Module 33: Review Question Answers

Answer True or False.

1. True or False. With BTEQ you can import data from the host to Teradata AND export from Teradata to the host.
2. True or False. .EXPORT DATA sends results to a host file in field mode. (*Results are in record mode.*)
3. True or False. INDICDATA is used to preserve nulls.
4. True or False. With BTEQ, you can use conditional logic to bypass statements based on a test of an error code.
5. True or False. It is useful to employ multiple sessions when ALL AMPS will be used for the transaction. (*It is useful when fewer than all AMPs are used.*)
6. True or False. With .EXPORT, you can have output converted to a format that can be used with PC programs.



Module 34: Review Question Answers

Match the item in the first column to a corresponding statement in the second column.

- | | |
|-----------------------------|--|
| 1. <u>C</u> Phase 1 | A. Might be used if a zero date causes an error |
| 2. <u>G</u> CHECKPOINT | B. Table status required for loading with FastLoad |
| 3. <u>H</u> ERRORTABLE1 | C. Records written in unsorted blocks |
| 4. <u>D</u> ERRORTABLE2 | D. Records rows with duplicate values for UPI |
| 5. <u>B</u> Empty Table | E. Not permitted on table to be loaded with FastLoad |
| 6. <u>E</u> Secondary Index | F. Points FastLoad to a record in an input file |
| 7. <u>J</u> Conversion | G. Can be used to restart loading from a given point |
| 8. <u>A</u> NULLIF | H. Records constraint violations |
| 9. <u>F</u> RECORD | I. Builds the actual table blocks for the new table |
| 10. <u>I</u> Phase 2 | J. Transform one data type to another, once per column |



Module 35: Review Question Answers

Match the item in the first column to its corresponding statement in the second column.

- | | |
|-----------------------|---|
| <u>D</u> 1. .LOGTABLE | A. Connects sessions to Teradata |
| <u>A</u> 2. .LOGON | B. Uses a single data record to set one or more utility variables |
| <u>B</u> 3. .ACCEPT | C. System variable |
| <u>E</u> 4. UPDATE | D. Identifies the log to create or acquire |
| <u>C</u> 5. &SYSDATE | E. Teradata SQL statement permitted by Support Environment |



Module 36: Review Question Answers

Answer True or False.

1. True or False. FastExport requires the use of a PI or USI in the SELECTs.
2. True or False. The number of FastExport sessions (for a UNIX server) defaults to the number of AMPs. (*The UNIX default is 4 sessions.*)
3. True or False. The maximum block size you can specify is 128 KB. (*It is 64 KB.*)
4. True or False. You can export from multiple tables with FastExport.
5. True or False. You can use multiple SELECTs in one FastExport job.
6. True or False. The default lock for a SELECT in a FastExport job is a table level ACCESS lock.



Module 37: Review Question Answers

Answer True or False.

1. True or False. With MultiLoad, you can import data from the host into populated tables.
2. True or False. MultiLoad cannot process tables with USIs or Referential Integrity defined.
3. True or False. MultiLoad allows changes to the value of a table's primary index.
4. True or False. MultiLoad allows you to change the value of a column based on its current value.
5. True or False. MultiLoad permits non-exclusive access to target tables from other users except during Application Phase.

Match the MultiLoad Phase in the first column to its corresponding task in the second column.

- | | |
|-----------------------------|---|
| 1. <u>A</u> Preliminary | A. Acquires or creates Restart Log Table. |
| 2. <u>E</u> DML Transaction | B. Locks are released. |
| 3. <u>C</u> Acquisition | C. Applies (loads) data to the work tables. |
| 4. <u>D</u> Application | D. Execute mload for each target table as a single multi-statement request. |
| 5. <u>B</u> Cleanup | E. Stores DML steps in work tables |



Module 38: Review Question Answers

1. Complete the BEGIN statement to accomplish the following:

- Specify an error limit count of 200,000 and an error percentage of 5%.
- Specify a checkpoint at 500,000 records.
- Request 16 sessions, but allow the job to run with only 8.
- Set the number of hours to try to establish connection as 6.

```
.LOGTABLE RestartLog_mld;
.LOGON _____;
.BEGIN [IMPORT] MLOAD TABLES Trans_Hist

    ERRLIMIT 200000 5
    CHECKPOINT 500000
    SESSIONS 16 8
    TENACITY 6
;

.END MLOAD ;
```



Module 39: Review Question Answers

Match the item in the first column to its corresponding statement in the second column.

- | | |
|-----------------------------------|---|
| <u>C</u> 1. TPump purpose | A. Query against TPump status table |
| <u>E</u> 2. MultiLoad purpose | B. Concurrent updates on same table |
| <u>B</u> 3. Row hash locking | C. Low-volume changes |
| <u>D</u> 4. PACK | D. Use to specify how many statements to put in a multi-statement request |
| <u>F</u> 5. MACRO | E. Large volume changes |
| <u>A</u> 6. Statement rate change | F. Used instead of DML |



Module 40: Various Ways of Performing an Update Solution

Answer: 2, 4, 3, 1 (Note: Timings are for an older and very small Teradata system).

- UPDATE (Do the UPDATE statement as shown):

Timings: Total time: 29 minutes and 9 seconds

- INSERT / SELECT the revised values into a new table; drop the old table and rename the new table.

Timings:	Create new table	3 seconds
	Insert>Select 900000 rows	1 minute 26 seconds
	Drop Old Table	11 seconds
	Rename New Table	1 second
	Total Time:	1 minute , 41 seconds

- .EXPORT the new data values and the primary index columns to the Host and use MultiLoad UPDATE.

Timings:	Export 900,000 rows	14 seconds
	MultiLoad/Update	3 minutes, 27 seconds
	Total Time:	3 minutes , 41 seconds

- .EXPORT the whole rows to the Host selecting the updated values along with the rest of the record, and FastLoad the table.

Timings:	Export 900,000 rows	16 seconds
	Delete old rows	6 seconds
	FastLoad the data	1 minute 59 seconds
	Total Time:	2 minutes, 21 seconds



Module 40: Choosing a Utility Exercise Solution

1. A sales table currently contains 24 months of transaction data. At the end of each month, 25 million rows are added for the current month and 25 million rows are removed for the oldest month. There is enough PERM space to hold 30 months worth of data.

Which choice (from below) makes the most sense? D

2. The customer decides to partition the table by month and maintain each month's data in a separate partition. At this time, only the most recent 24 months need to be maintained. At the end of each month, data is loaded into a new monthly partition and the oldest month is removed.

Which choice (from below) makes the most sense? B

Utility Choices:

- A. Use FastLoad to add new data to existing table, and ALTER TABLE to remove old data.
- B. Use MultiLoad to add new data to existing table, and ALTER TABLE to remove old data.
- C. Use FastLoad to add new data to existing table, and MultiLoad to remove old data.
- D. Use MultiLoad to add new data to existing table, and MultiLoad to remove old data.
- E. Use TPump to add new data, and TPump to remove old data.
- F. Use TPump to add new data, and ALTER TABLE to remove old data.



Module 41: Review Question Answers

1. **True or False.** You should use system user DBC to create application users and databases.
False. You should create and logon as an administrative user to perform these tasks.
2. **True or False.** An database or user can have multiple Owners, but only one Creator.
3. **True or False.** A Parent and an Owner are two different terms that mean the same thing.
4. **True or False.** A Parent and a Creator are two different terms that mean the same thing.
The term "Creator" is used to mean the one and only one user who creates a database object.
5. **True or False.** An administrative user (e.g. Sysdba) will never have more permanent space than DBC.
Sysdba can be allocated more permanent space than DBC.
6. **True or False.** The GIVE statement transfers a database or user space to a recipient you specify. It does not automatically transfer all child databases of the transferred database or user.
False. The GIVE statement automatically transfers all child databases, users, tables, view and macros the transferred object owns.



Module 42: Review Question Answers

1. True or False. The DBC.Databases view only contains information about databases; users are not included in this view.
2. True or False. The DBC.Users view only contains information about users; databases are not included in this view.
3. True or False. Queries that use restricted views usually take less time to execute than queries that use unrestricted views.
4. True or False. All of the data dictionary tables are Fallback protected.
5. If a child table exists and the parent table doesn't, the reference constraint is marked as _____.
 - A. Inconsistent
 - B. Unresolved
 - C. Missing
 - D. Invalid
6. After executing the ALTER TABLE ... ADD FOREIGN KEY ... statement, Foreign Key values that are missing in the parent table are marked in an error table and are known as _____ rows.
 - A. Inconsistent
 - B. Unresolved
 - C. Missing
 - D. Invalid



Module 43: Review Question Answers

1. True or False. Space limits are enforced at the table level.
2. True or False. When you use the GIVE statement to transfer a database/user, only the tables allocated to the original database/user are transferred to the new database/user.
3. True or False. You should reserve anywhere from 20 - 25% of total available space for spool.
4. The DBC.Tablesize view provides disk space usage at the table level and excludes table ALL.
5. The DBC.DiskSpace view only provides disk space usage at the database level.



Module 44: Review Question Answers

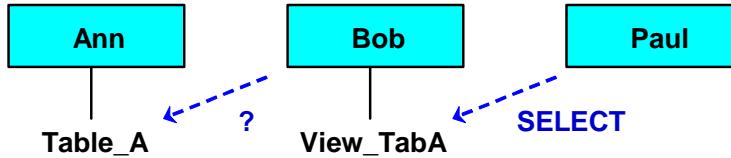
1. True or False. You can only give the authority to use the CREATE DATABASE and CREATE USER statements to certain types of users.
2. True or False. An individual user with a \$L priority will always receive less CPU time than a user with a \$M priority.
3. True or False. A user can use the MODIFY USER statement to change their password, default database, and date format.
4. When creating a new user, which of the following option(s) defaults to the immediate owner's value. A
 - A. SPOOL
 - B. FALBACK PROTECTION
 - C. All of the Account_IDs
 - D. DEFAULT DATABASE
5. When creating a new user, which of the following option(s) are required with the CREATE USER command. B, C, D
 - A. SPOOL
 - B. PERMANENT
 - C. User name
 - D. PASSWORD

Module 45: Review Question Answers

1. True or [False](#) There are only two types of access rights or privileges: explicit and implicit.
2. True or [False](#) The primary statements you use to manage access rights are GRANT, REVOKE, and GIVE.
3. The [ALL](#) option on the GRANT command grants privileges to a database or user and all of its current and future descendants.
4. The [UPDATE](#) and [REFERENCES](#) access rights can be granted at the column level.
5. The [PUBLIC](#) user is used to grant an access right to every user in the system.
6. Given the following: Ann owns Table_A, Bob creates View_TabA and grants SELECT on View_TabA to Paul.

What access right does Ann give Bob on Table_A so Paul can use View_TabA to access Table_A?

[SELECT WITH GRANT OPTION](#)





Module 46: Review Question Answers

Answer the following questions:

1. List 3 advantages of utilizing roles and profiles.

Simplify access rights management

The number of access rights in the DBC.AccessRights table is reduced.

Improves performance and reduces dictionary contention

2. How many levels of role nesting are currently allowed? 1

3. True or False. A user can use the SET ROLE command to set their current role to any defined role in the system.

4. True or False. Roles may only be granted to users and other roles.

Match each term to the definition.

- | | | |
|---|-----------------------|---|
| E | 1. WITH ADMIN OPTION | A. Established by the SET ROLE command |
| C | 2. CREATE ROLE | B. Lists the roles currently available to a user |
| B | 3. DBC.RoleInfo | C. System access right needed to create a role |
| D | 4. DBC.UserRoleRights | D. Lists all of a user's role rights - including nested roles |
| F | 5. DEFAULT ROLE | E. Allows the user to assign other users to the role |
| A | 6. Current role | F. Option with the MODIFY USER statement |



Module 47: Review Question Answers

1. Given the following, what is minimum % of resources that the following Performance Groups (PG) can expect.

$$\text{TAC_M} = \underline{6\% \ (60/100 \times 10/100 = .06)}$$

$$\text{DSS_H} = \underline{5\% \ (20/100 \times 30/120 = .05)}$$

Default		Tactical		Standard	
<u>RP 0 - Weight 20</u>		<u>RP 1 - Weight 60</u>		<u>RP 2 - Weight 20</u>	
<u>PG Name</u>	<u>Wgt</u>	<u>PG Name</u>	<u>Wgt</u>	<u>PG Name</u>	<u>Wgt</u>
L	5	TAC_L	5	DSS_L	5
M	10	TAC_M	10	DSS_M	10
H	20	TAC_H	30	DSS_H	30
R	40	TAC_R	55	DSS_R	75

2. Without TASM workloads enabled, a user session is associated with a Performance Group which is effectively assigned to an Allocation Group.
3. With TASM workloads enabled, a user query is associated with a Workload which is effectively assigned to an Allocation Group.



Module 48: Review Question Answers

1. True or False. Although usernames are the basis for identification in the system, username information usually is not protected information.
2. True or False. Once users have a username and password, they can access any information in the system.
3. True or False. If you want to change the minimum number of characters in a valid password from 4 to 6, you would update the DBC.LogonRules table.
4. What does 1024 represent in the DBC.LogonRules view? All of the hosts
5. Which of the following choice(s) can be used to view host or mainframe sessions? A, C
 - A. QrySessn utility
 - B. Sessions utility
 - C. DBC.SessionInfo view
 - D. Gtwglobal utility
6. Which one of the following choices is used to determine why a user logon has failed? E
 - A. DBC.Logons view
 - B. DBC.AccessLog view
 - C. DBC.LogonEvents view
 - D. DBC.SessionInfo view
 - E. DBC.LogOnOff view



Module 49: Review Question Answers

1. In order to use the BEGIN/END LOGGING commands, what is the name of the system macro you need execute permission on?

DBC.AccLogRule

2. How is this macro initially created?

When the DIP script (DIPACC) is executed.

3. What is a negative impact of the following statement?

BEGIN LOGGING WITH TEXT ON EACH

Potentially a lot of entries are placed in the dictionary and would require a lot of PERM space.

4. With DBQL, what is the size of the default text captured for queries? 200 characters
5. True or False. With DBQL, the LIMIT SUMMARY option cannot be used with any other LIMIT.
6. True or False. With DBQL, the WITH SQL option only captures a maximum of 10,000 characters.
7. True or False. With DBQL, the option WITH ALL ON ALL is typically a good choice.
8. True or False. With DBQL, default rows are logged in the DBC.DBQLogTbl.



Module 50: Review Question Answers

1. What type of TDWM filter or throttle rule is needed for the following restrictions?

Limit the number of concurrent sessions

Object Throttle

Reject queries based on max processing time

Query Resource Filter

Reject queries accessing a specific DB

Object Access Filter

Limit the number of FastLoad jobs

Load Utility Throttle

Delay more than 20 queries for a specific account

Object Throttle

2. What is the purpose of the default workload definition name "WD-Default"?

- A. Default workload for any queries with an enforcement policy of normal.
- B. Default workload for any queries that are not associated with a workload.
- C. Default workload for any queries assigned to Default resource partition.
- D. Default workload for any queries assigned to Standard resource partition.

3. Which query attribute is not used by the Parsing Engine software to 'classify' a query into a Workload Definition (WD)?

- A. User name
- B. Account
- C. User Role
- D. User Profile
- E. Optimizer estimates



Module 50: Review Question Answers (cont.)

4. Which Teradata application is used to activate a workload definition rule set?
 - A. Workload Analyzer
 - B. Priority Scheduler Administrator
 - C. Teradata Manager
 - D. Dynamic Workload Manager
5. Which exception action is NOT possible?
 - A. Do nothing and simply log the exception
 - B. Delay the query to off-hours
 - C. Abort the query
 - D. Send an alert to the DBA
6. Which criteria has less overhead?
 - A. Who criteria
 - B. What criteria
 - C. Where criteria
 - D. All of these have similar overhead
7. Place the following control options in the proper sequence from 1 to 4 as they are acted upon by Teradata software.
 - 2 A. Object throttles
 - 4 B. Exception criteria
 - 3 C. Workload throttles
 - 1 D. Object filters



Module 51: Review Question Answers

1. List three tools of Teradata Manager.

[Remote Database Console](#)
[Teradata Performance Monitor](#)
[Locking Logger](#)



Module 52: Review Question Answers

1. What are two methods of setting collection and logging intervals?

[Supervisor commands](#)

[xctl, ctl, or MultiTool CTL utility](#)

2. Match the following tools to its description.

- | | |
|--|---------------------------------------|
| <u>D</u> 1. ResUsage tables | A. Set collection and log rates |
| <u>C</u> 2. Teradata Performance Monitor | B. Emulates a target system |
| <u>B</u> 3. Teradata SET | C. Provides session level information |
| <u>A</u> 4. ctl | D. Holds historical resource data |



Module 53: Review Question Answers

1. What are four ways that you initiate an Teradata system utility (e.g., dbscontrol)?

[Teradata DB Window](#) [Command-line](#) [MultiTool](#) [TM Remote Console](#)

2. Identify the purpose of the following DBS Control utility parameters.

CenturyBreak	<u>determines break point for 21st century</u>
DateForm	<u>sets IntegerDate or ANSIDate default</u>
DictionaryCacheSize	<u>amount of DD cache memory to allocate per parsing engine</u>
MaxLoadTasks	<u>maximum number of concurrent FastLoad, MultiLoad, and FastExport jobs</u>
PermDBSize	<u>sets blocksize default</u>
SessionMode	<u>sets default session mode – BTET or ANSI</u>



Module 54: Review Question Answers

1. What is the operating system command to restart Teradata? [tpareset](#)
2. What is the DB Window supervisor command to restart Teradata? [restart tpa](#)
3. Which of the following choices will cause a Teradata restart? [C, E](#)
 - A. AWS hard drive failure
 - B. Single drive failure in RAID 1 drive group
 - C. Two drive failures in same RAID 1 drive group
 - D. Single SMP power supply failure
 - E. SMP CPU failure
 - F. One of BYNETs fails
 - G. LAN connection to SMP is lost



Module 55: Review Question Answers

1. True or [False](#). The Checktable utility features two levels of internal table checking.
2. [True](#) or False. The Table Rebuild utility rebuilds tables differently depending on whether the table is a fallback, non-fallback or permanent journal table.
3. The [SCANDISK](#) utility does a consistency check within an AMP's file system.
4. The [CHECKTABLE](#) utility does a consistency check for a table across all AMPs.
5. The [VPROCMANAGER](#) utility can be used to set an offline AMP to online.



Module 56: Review Question Answers

1. True or False. A permanent journal stores committed, uncommitted, and aborted changes to a row in a table.
2. True or False. A database or user can have many permanent journals.
3. True or False. Separate Permanent Journals are required for before and after images.
4. True or False. The Saved and Active areas are both part of the Current Journal.
5. True or False. The CREATE JOURNAL statement is used to create a permanent journal.
6. True or False. Tables that use a Permanent Journal must be in the same database as the Permanent Journal.



Module 58: Review Question Answers

1. True or False. The Archive and Recovery utility protects against more types of potential data loss than automatic data protection features.
2. True or False. Recovery and FastLoad are about the same in ease and speed to recover data.
3. True or False. An All-AMPs archive of a database archives all of the objects in the database.
4. True or False. Archiving a partition of a PPI table places a partition-level lock on the partition being archived.



Module 59: Review Question Answers

1. True or False. You can use the RESTORE command to restore entities that are not defined in the data dictionary.
2. True or False. When you execute a RESTORE of a database, any tables or views created since the archive of the database are dropped when you restore the database.
3. True or False. You can use the COPY operation to copy tables, views, macros, and triggers from one system to another system.
4. The REVALIDATE REFERENCES FOR statement is used to validate Referential Integrity between tables that are identified as _____.
 A. Inconsistent
B. Unresolved
C. Missing
D. Invalid



Module 60: Review Question Answers

1. True or [False](#). The DELETE JOURNAL command can be used to delete the active and the saved areas of the current journal.
2. In general, rollback operations help you recover from [software](#) failures and rollforward operations help you recover from [hardware](#) failures.
3. To use the ARCHIVE JOURNAL TABLE command to archive a permanent journal, the active journal images need to be moved to the saved area of the current journal. The command to do this is:

[CHECKPOINT ,WITH SAVE](#)

Teradata Training

Notes

Module E



Appendix E: TQS and TWA

This Appendix contains additional information about the Teradata Query Scheduler, Teradata Workload Analyzer, and Teradata Manager enhancements for TASM.

Teradata Proprietary and Confidential

Teradata Training

Notes

Table of Contents

Teradata Query Scheduler	E-4
Scheduling Queries	E-4
Teradata QS – Architecture for Scheduled Requests	E-6
Administration Components	E-6
Server Components	E-6
Client Components	E-6
Teradata QS Administration – Profiles	E-8
Configuring Scheduler Profiles	E-8
Saving Result Data	E-8
Workgroups, File Storage, and Time Frames	E-10
Setting up Workgroups	E-10
Managing Results File Storage	E-10
Configuring Execution Time Frames	E-10
Teradata QS Viewer – Scheduling a Request	E-12
Saving Result Data	E-12
Teradata SQL Assistant – Scheduling a Request	E-14
Enabling Scheduling of Rejected Teradata SQL Assistant Queries	E-14
Teradata Workload Analyzer	E-16
Step 1 – Collect Data for Workload Analyzer	E-18
Step 2A – Select PD Set(s) and DBQL	E-20
Step 2B – Select Performance Groups	E-22
Step 2C – Convert PD Sets to Workload Definitions (Remap As-is)	E-24
Step 2C – Convert PD Sets to Workload Definitions (Simplified)	E-26
Teradata Manager Enhancements	E-30
Teradata Manager – Dashboard Enhancements	E-32
Dashboard – Workload Snapshot	E-34
Dashboard – Workload History	E-36
Dashboard – Workload Summary History	E-38
Recommendations on Features to Use	E-44

Teradata Query Scheduler

Teradata Query Scheduler provides a Scheduled Requests capability to the Teradata Database. Scheduled Requests are user requests to submit queries for off-line execution at later or more preferable dates and times. This does not; however, guarantee that scheduled requests will be executed at the deferred time. Therefore the user must also supply a time period after the scheduled time that the request can be executed.

Scheduling Queries

The Teradata Query Scheduler (QS) server attempts to execute requests during the time period specified, but **restrictions still govern the execution of scheduled requests**. When end-users know of existing database rules that will prevent a SQL request from running or if they suspect their queries will overload the Teradata Database, they can actively schedule a request using the Teradata Query Scheduler Submit dialog box.

Teradata QS monitors SQL requests actively scheduled using the Teradata Query Scheduler Submit dialog box and requests scheduled because of database workload management. Teradata QS manages workloads submitted to your Teradata Database by executing the request immediately or suspending the request to run at a later time. By actively managing your Teradata Database, Teradata QS makes sure that the most important work gets done at the best time and that system resources are not overloaded.

When a request is scheduled, end-users provide information that defines preferences for when it is executed. A request can be scheduled to run periodically or only once during a specified time period without an active system user connection.

Because a scheduled request can actually be executed many times, the term **request** is used to mean the actual definition of the scheduled request parameters. The term **job** is used to mean an individual instance a scheduled request is scheduled to run.

For example, a scheduled **request** can be defined to execute *daily*. That request causes a separate **job** to be created *every day* to execute that request.

When a request is submitted for scheduling by the user, the request is analyzed prior to acceptance by QS. If one or more access and/or query resource restrictions would keep the scheduled request from running over the entire time period in which it may be executed, then the request will fail. However, if there is a later time within the specified execution interval when it is not restricted, then the scheduled request will be rescheduled for some time later.

Typically, the Teradata QS client software is installed on many systems while the Teradata QS server software is installed on one or a limited number of servers. Administration of the Teradata QS capabilities is performed by a Teradata QS Administrator application.



Teradata Query Scheduler

The Teradata Query Scheduler provides a **Scheduled Requests** capability.

- Scheduled requests are SQL queries scheduled by end-users for off-line execution.
- What are the main components?
 - **TQS Server components** – software that runs on a Windows-based server that monitors the job to determine when it can be run and submits the job.
 - **TQS Client components**
 - **TQS Viewer** – to manage the request results
 - **TQS Submit Dialog Box** – submit single or multiple SQL statement requests for execution at a later date and time
 - **TQS Operations Utility** – tool to configure and start the TQS server
 - **TQS Administrator** – program to enable “scheduled requests”, set up user profiles, and setup time frames in which schedule requests are executed by TQS.
- Are there any other user applications that can also automatically schedule requests?
 - Teradata SQL Assistant does it by recognizing the DWM error codes that come from the database when a query is delayed because of a rule violation.

Teradata QS – Architecture for Scheduled Requests

Administration Components

The administrator components include:

- Teradata Query Scheduler Administrator
- Teradata Query Scheduler Setup utility

You use the Teradata Query Scheduler Administrator to enable the scheduled request feature, set up user profiles, and setup time frames in which schedule requests are executed by QS.

You also run the Teradata Query Scheduler Setup utility to migrate earlier versions of the QS database (**dbqrymgr**) to the new version called **tdwm**. Scheduled request information is stored by QS in **tdwm**.

Server Components

The server component includes:

- Teradata QS scheduler/dispatcher
- Teradata QS executor program
- Teradata QS request processors
- Teradata QS communications library

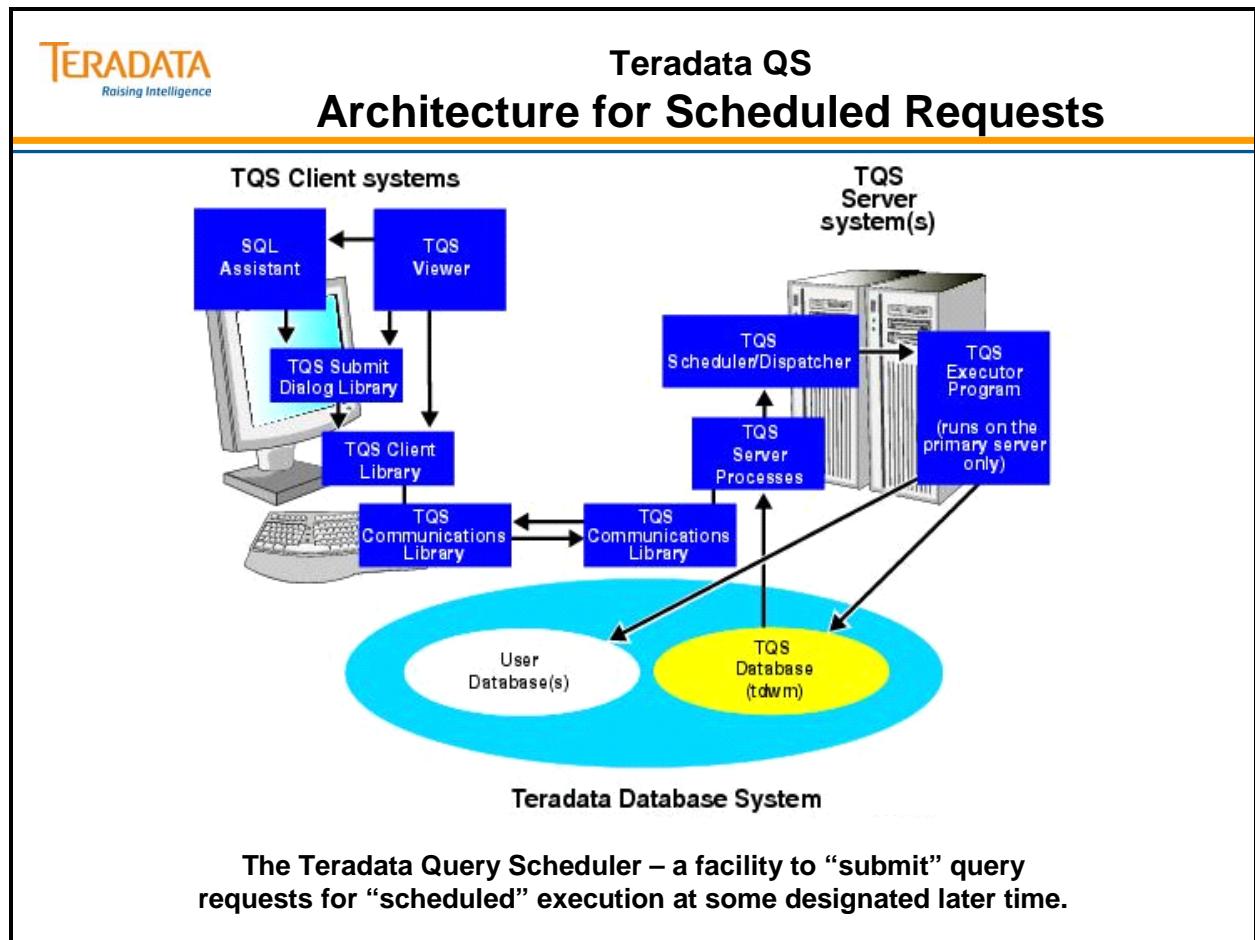
The QS server components execute as a Windows application. The purpose of the server is to save, process, and execute QS client requests that have been scheduled.

Client Components

Since the Teradata database does not currently schedule queries automatically, it must be done by a “client” application. Two ways to access the Teradata Query Scheduler Submit dialog box are via TQS Viewer or Teradata SQL Assistant. Teradata SQL Assistant does it by recognizing the DWM Query Management error codes that come from the database when a query fails, and performs the necessary steps to invoke the Teradata Query Scheduler Submit dialog box for collecting the scheduling information and submitting the scheduled request.

To schedule a query that is rejected because of DWM rules, you must use Teradata SQL Assistant, not BTEQ.

The results of Scheduled Requests that have been executed can be retrieved using any ODBC program if the results were saved in a database table. However if the results were saved in a “server” file, then they can only be retrieved using the Teradata QS Viewer application.



Teradata QS Administration – Profiles

The scheduled requests feature allows any user to submit SQL requests for scheduled execution. You must enable the feature on your Teradata Database system before SQL requests can be scheduled or processed.

- To enable scheduled requests, use the Configuration menu, select Enable Scheduling.

Configuring Scheduler Profiles

You use profiles to control scheduled requests submitted to your QS server. A profile is a set of parameters you assign to a individual users, accounts, DBS Roles, or DBS Profiles that determines what scheduling capabilities are available and how your QS server handles their scheduled requests.

A default user profile is created and reserved by QS during installation and is used when a user is not assigned to a user profile, a group profile, or an account profile. You cannot delete the default user profile, but can change it.

Saving Result Data

Most scheduled requests, especially automatically scheduled requests, are not self-contained; that is, they do not specify where to save the results data. You can have QS save the results for those types of requests. However, results from only the last statement in the scheduled request are saved.

To do this, the last statement in your scheduled SQL request must be a **SELECT** statement, and you must specify a valid Teradata Database and table or a valid QS server file name in which to store the results.

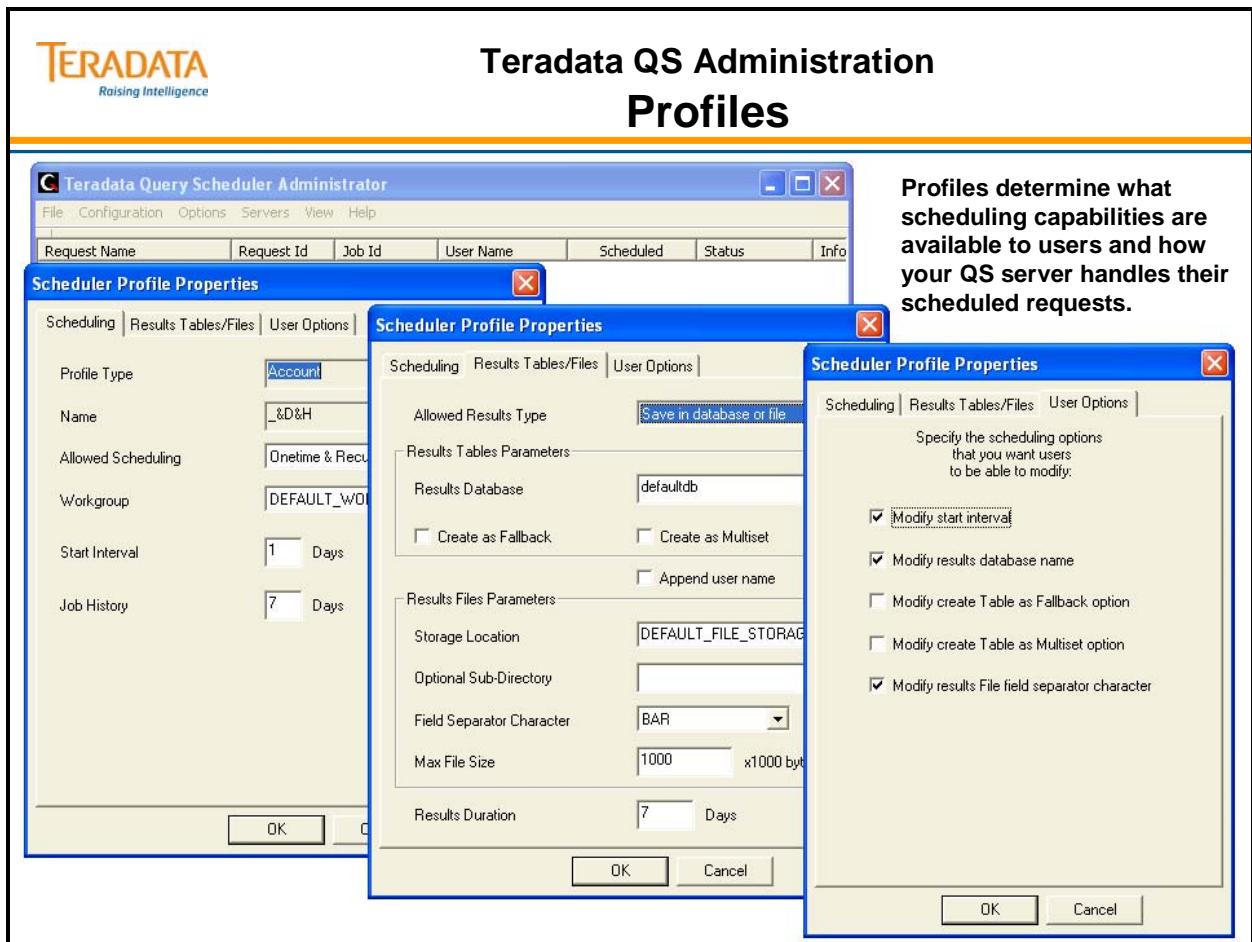
If the results table or file already exist when the scheduled request is run, one of the following events occurs:

- The job execution is aborted and the job is marked as failed.
- The results table is destroyed, if so specified. Then, the results table is recreated with the proper column types and the data from the last statement is stored.

Which event occurs depends on your selections in the Results tab of the Teradata Query Scheduler Submit Request dialog box (shown later in this module).

If the Append User Name option is checked, the user's name is appended to the result table or file. For example: Result table name is: Orders_GT_500tfact03 (user was tfact03)

The Teradata QS Administrator allows you to specify defaults for results storage as shown on the facing page.



Workgroups, File Storage, and Time Frames

Setting up Workgroups

A workgroup represents a collection of related scheduled requests for a number of users, accounts, DBS Roles, and DBS Profiles. You can create, edit, or delete any number of workgroups that equitably distribute resources and scheduled requests workloads during execution time frames.

You assign a maximum number of requests that can be simultaneously executing from each workgroup. This ensures that a fair share of the scheduled work gets done within the execution time frames for all workgroups.

Managing Results File Storage

If the last or only, statement in a scheduled request is a SELECT statement, it generates a result set that can be captured and saved by QS. Results from a scheduled request can be saved to a new or existing database table, or to a file specified when the request was created.

Results file storage is a symbolic name for a file system directory where scheduled requests results files are stored. If you want the results of scheduled requests to be saved in “flat text” files, you must designate the storage location. You can create, edit, or delete any number of locations where scheduled request results files are stored and accessed by all QS servers.

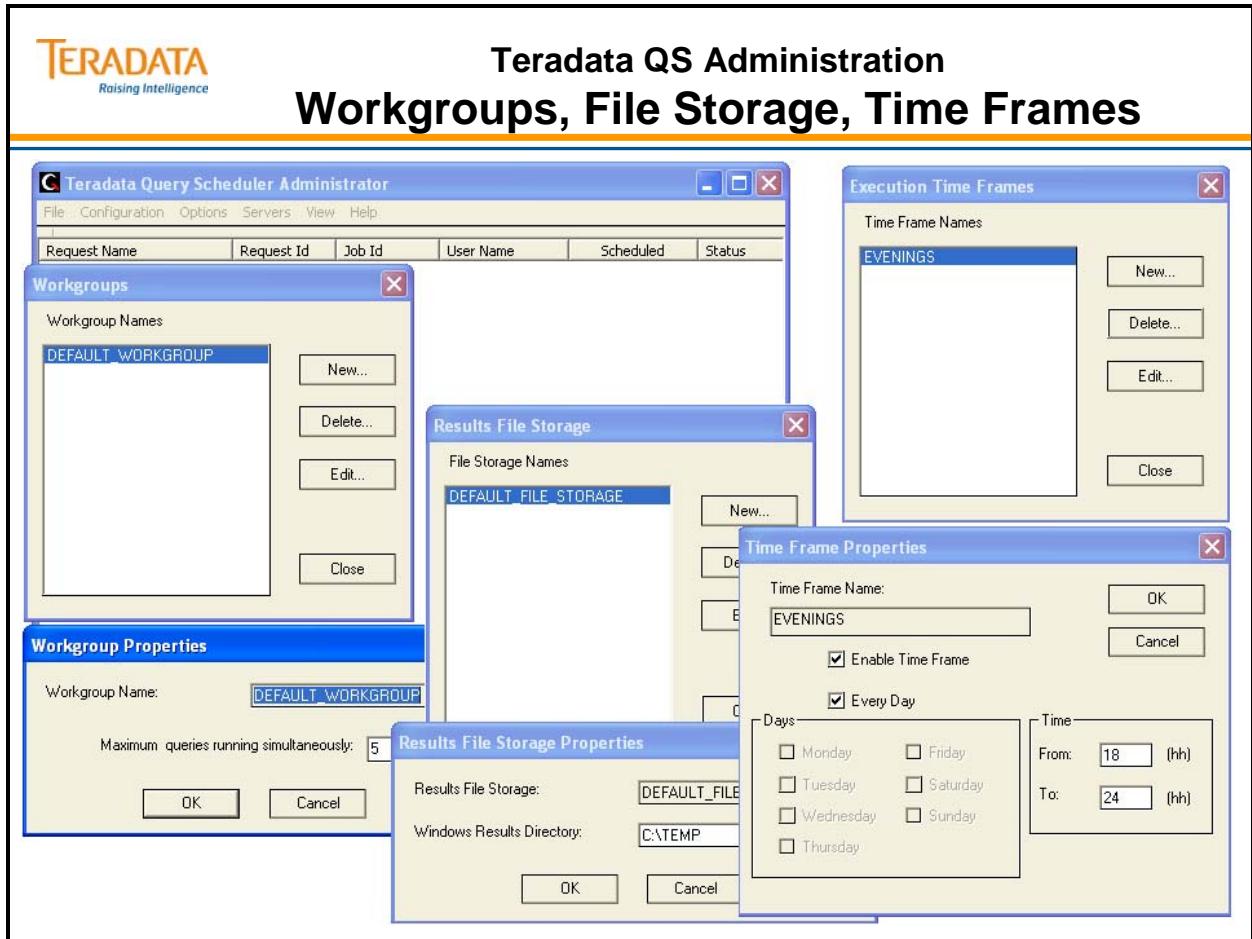
Each results file storage location has a specified *Windows* path. The same *Windows* path is used by all active QS servers so that they can all access the results files at the same location. This requires that all QS scheduled requests servers must have access to the Windows path(s) that describe the results file storage location(s). Profiles additionally contain an optional relative path that is used to reference a subdirectory name for the desired profile.

Configuring Execution Time Frames

You control when scheduled SQL requests are executed on your system by defining and enabling execution time frames. An execution time frame is a period of time in which scheduled SQL requests are permitted to run and when the QS server will attempt to execute any scheduled requests.

After you create an execution time frame, you must make it available on your Teradata Database system. Scheduled requests are not accepted or run unless an execution time frame is available. When a user submits a request for scheduling, QS determines whether there are any execution time frames available during the scheduled time period when query management rules do not apply. If no execution time frames are available and the maximum start interval is exceeded, then the request is not scheduled.

Although you can create and enable multiple execution time frames, only the primary QS server actually dispatches and executes the scheduled requests.



Teradata QS Viewer – Scheduling a Request

The Teradata Query Scheduler Submit dialog box is used to submit a scheduled request to Teradata Query Scheduler (QS). This dialog box is accessed either from the Teradata QS Viewer or Teradata SQL Assistant applications.

Using the Teradata QS Viewer client application, the Command > Schedule SQL menu options open the SQL Text box. In this box, enter the text for the single or multiple SQL statements that you want to schedule.

Note: You cannot use scheduled requests to insert large object (LOB) data into a Teradata Database. However, you can schedule a SQL request that retrieves LOB data from a table, and save those results to a table or a file.

The scheduling information you provide is used by QS as the intended start time for the request. The QS server attempts to execute requests during the time period specified, but Teradata Database workload management rules determine if or when the scheduled request is executed.

Because a request may not be run until some time after the requested start time, you must also specify a time interval in which the request may be run. Submitted requests are immediately rejected if an execution time frame is not available during the specified interval.

By using the Teradata Query Scheduler Administrator application, the DBA controls when scheduled requests are executed. The DBA specifies one or more execution time frames, or time periods, in which scheduled requests are executed by the QS server.

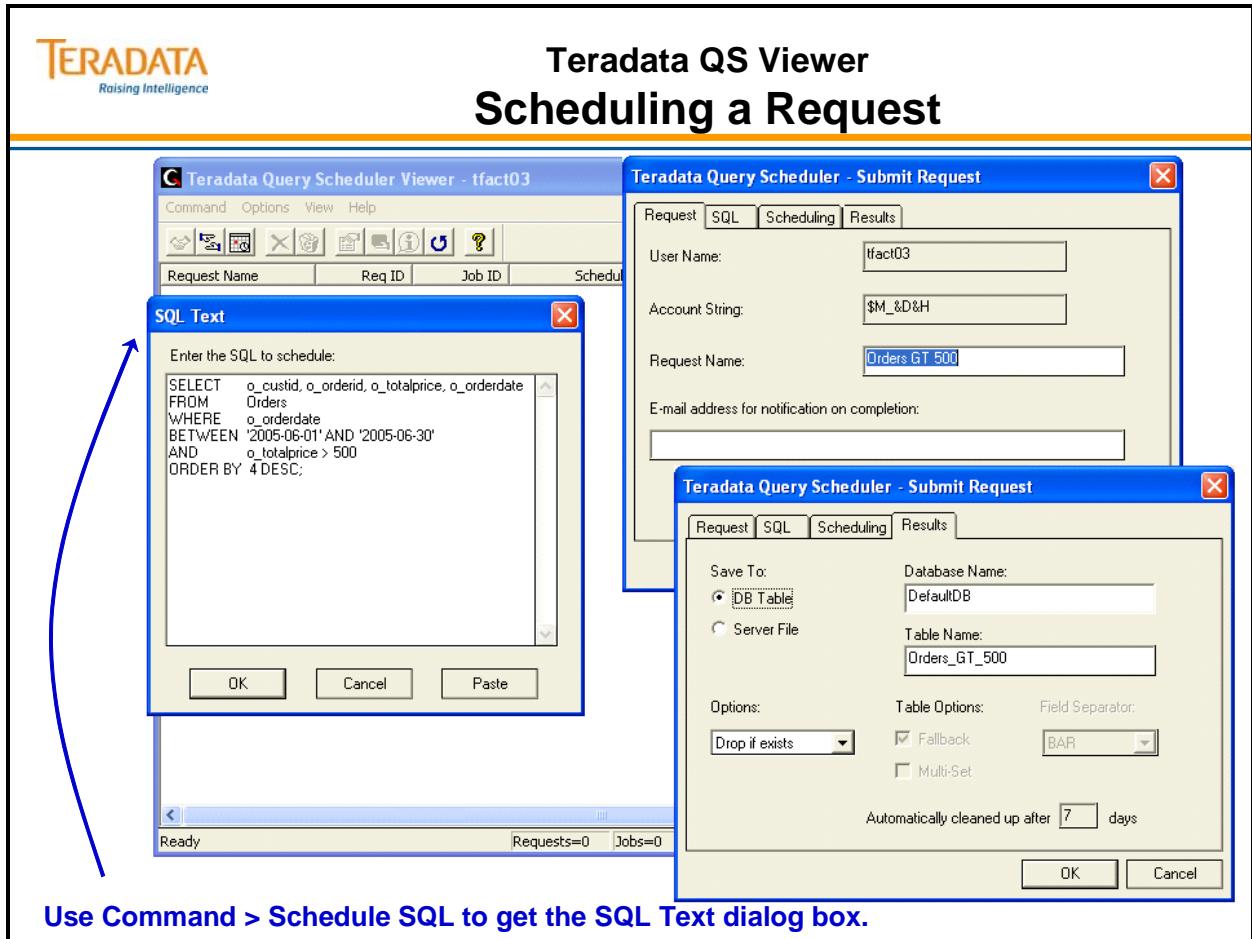
Your DBA must specify active execution time frames. If execution time frames are not defined or enabled, scheduled requests are not executed by Teradata QS.

Depending on how a user's profile is created (capabilities that are allowed), the user may (or may not) be able to modify different scheduling and/or results options.

Saving Result Data

The Results tab allows you to choose storage options for the results of a query. Choose how the results are handled when the database table already exists using one of these options:

- *Append if exists*: Appended to an existing table
- *Drop if exists*: Stored after dropping the existing table. This option is the default.
- *Fail if exists*: Not stored if scheduled request failed because the result table already exists.



Teradata SQL Assistant – Scheduling a Request

The Teradata Query Scheduler Submit dialog box may optionally be available to end-users of Teradata SQL Assistant to submit a scheduled request to Teradata Query Scheduler (QS).

Enabling Scheduling of Rejected Teradata SQL Assistant Queries

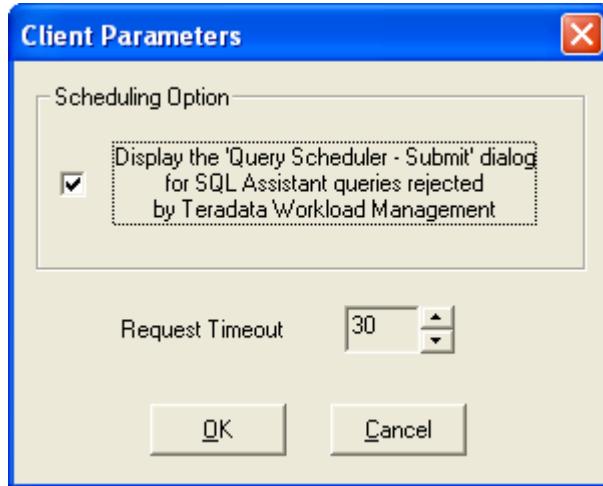
End-users can submit queries to the Teradata Database using Teradata SQL Assistant only if the option has been enabled.

When a Teradata SQL Assistant submitted query is rejected because of Teradata Database management rules, a status of 3149 or 3150 is returned to SQL Assistant. You can specify that the Teradata Query Scheduler Submit dialog box opens so that the end-user can schedule the rejected query to run at a later time.

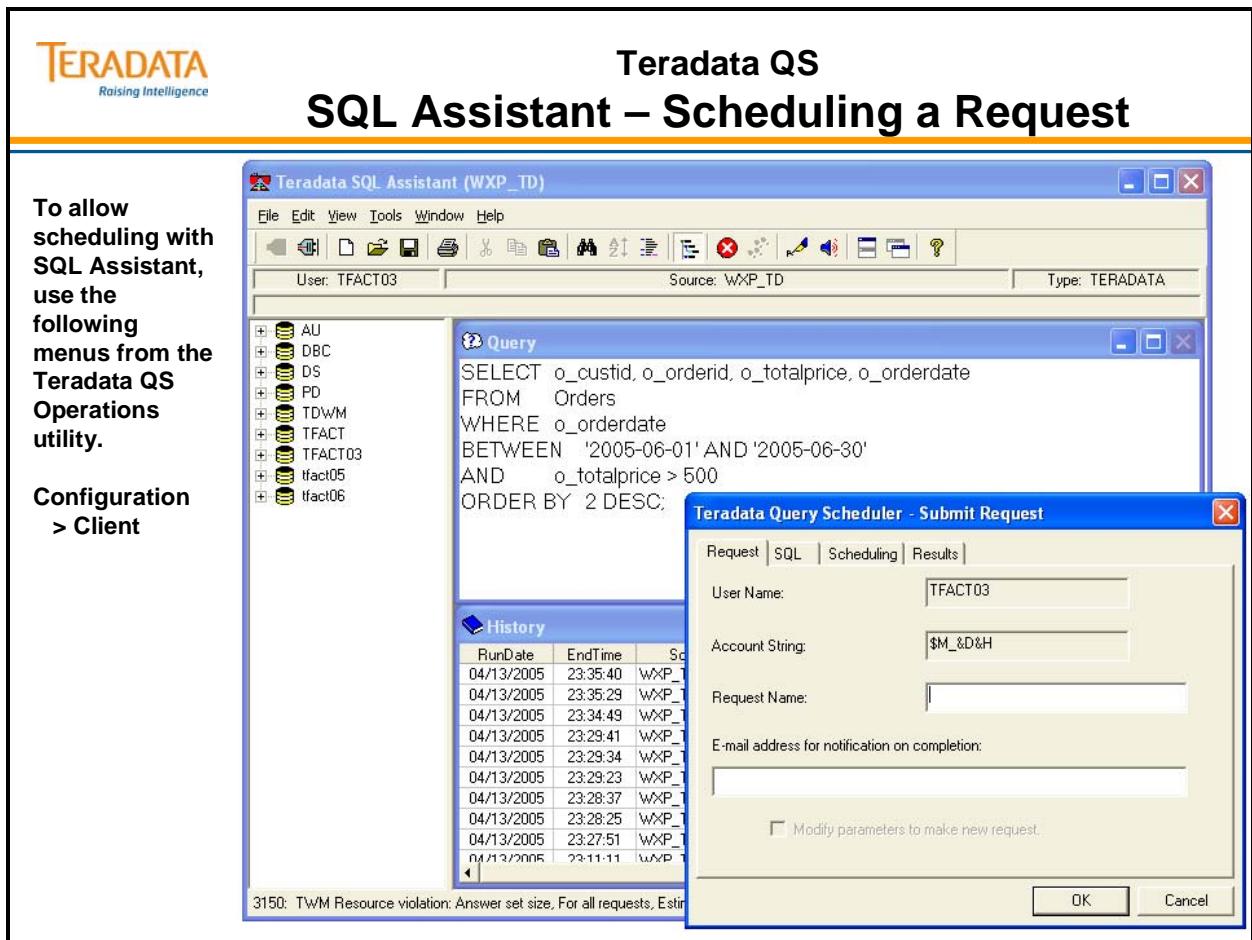
If you do not set up this option, the end-user does not have the opportunity to schedule the query using the Teradata Query Scheduler Submit dialog box.

To allow Teradata SQL Assistant to utilize this capability, use the **Teradata Query Scheduler Operations** utility to enable the Teradata SQL Assistant scheduling option.

To access the following dialog box, choose **Configuration > Client**.



Use the Request Timeout spin box to specify the number of seconds (between 5 and 99) to wait before timing out a request to the QS server. The default is 30 seconds.



Teradata Workload Analyzer

This application provides the following capabilities:

- Migrate existing PSF settings into workload definitions.
- Establish workload definitions from query history or directly
- Can be used “iteratively” to analyze and understand how well the existing workload definitions are working, and modify those definitions if necessary.

This tool combines data from existing Priority Scheduler settings (via Priority Definition or PD sets) and workloads (via Teradata DBQL – Database Query Log) to determine workload definitions for TDWM.

This application can also apply best practice standards to workload definitions such as assistance in SLG definition and priority scheduler setting recommendations.

In addition, Workload Analyzer supports the conversion of existing Priority Scheduler Definitions (PD Sets) into new workloads. A PD set is the collection of data, including the resource partition, allocation group, period type, and other definitions that control how the Priority Scheduler manages and schedules session execution.



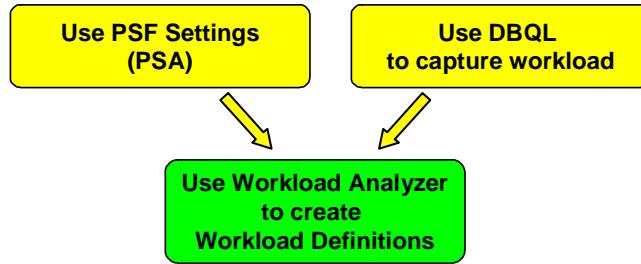
Teradata Workload Analyzer

Capabilities of Teradata Workload Analyzer include:

- Identifies classes of queries and recommends workload definitions
- Provides the ability to migrate existing PSF settings
- Recommends workload mappings and weights
- Provides recommendations for appropriate Service Level Goals (SLG)

Workload Analyzer combines data from two sources:

1. Existing Priority Scheduler settings (via Priority Definition or PD sets)
2. Workloads (via Teradata DBQL – Database Query Log)



Step 1 – Collect Data for Workload Analyzer

The migration process of Workload Analyzer combines information from two sources to initially recommend Workload Definitions. The two sources used by Workload Analyzer are:

- PD Sets – contain Priority Scheduler settings
- DBQL – contains workload activity that is generated by capturing queries

Priority Scheduler Settings

If you already use Priority Scheduler Administrator (PSA) to manage your Priority Scheduler settings, then you can skip this step.

To capture your current Priority Scheduler settings into a “PD Set” (Priority Definition), do the following:

1. Execute Teradata Manager.
2. Start Priority Scheduler Administrator from Teradata Manager.
Menus: Administer → Workload Management → Priority Scheduler
3. In Priority Scheduler Administrator, load your current Priority Scheduler settings.
Use the PSA menus: PDSet → Fetch
4. In the view window labeled **PD Set/Resource Partitions**, give the PD set a descriptive name in the text box labeled *PD Set*.
5. Save your new PD set to the database.
Use the PSA menus: PDSet → Save to DB.
6. If you use more than one set of PSF settings (e.g. one configuration during the day and a different configuration at night), then repeat steps 1-5 during the time that each PSF configuration is active. Write down the time periods that each PSF configuration is in effect – this information will be used later.

Create DBQL workload activity

If you are already collecting DBQL data for your workloads, then you can skip this step.

The Workload Analyzer will analyze both the DBQL detail table and the DBQL summary table. The detail table will contain one row for each query submitted. The summary table is much more efficient; it is updated at periodic intervals with a count of the number of queries submitted by each user or account. The Workload Analyzer does not look at the SQL text, so it is not necessary to capture the full SQL text.

To enable detail logging for a specific user or account use one of these commands:

```
BEGIN QUERY LOGGING ON USER1, USER2, USER3;  
BEGIN QUERY LOGGING ON ALL ACCOUNT = ('ACCT1', 'ACCT2', 'ACCT3');
```

An example to enable summary logging follows:

```
BEGIN QUERY LOGGING LIMIT THRESHOLD = 3600 ON USER1, USER2;
```

TERADATA
Raising Intelligence

Step 1 – Collect Data for Workload Analyzer

1. Use PSA to create PD Sets from existing Priority Scheduler settings.

2. Capture workloads via DBQL.

The screenshot shows the 'Priority Scheduler Administrator' window with the title '[TD61 - NewPDSet:Default]'. The main area displays a table for 'Performance Groups' with four rows (L, M, H, R) all set to 'Query' type and '0.00' milestone limit. To the right, the 'Resource Partitions' section shows 'Default' and 'Tactical' partitions with weights 100 and 200 respectively. Below that is the 'Allocation Groups' section, which lists five groups (AG1-AG5) with their details:

Name	ID	Resource Partition	Set Type	CPU Limit	Weight	Used by Perf G
AG1	1	Default	None	100	5	L(0)
AG2	2	Default	None	100	10	M(0)
AG3	3	Default	None	100	20	H(0)
AG4	4	Default	None	100	40	R(0)
AG5	5	Tactical	None	100	5	T_L1(1)

Step 2A – Select PD Set(s) and DBQL

The migration process of Workload Analyzer uses the DBQL log tables and Priority Scheduler settings (via PD Sets) to recommend a set of Workload Definitions. The Workload Analyzer determines how system resources are divided among the various workloads and how system resources are allocated to optionally meet Service-Level Goals (SLGs).

In this step, you will select the PD Sets that will be used to generate your candidate workloads. Depending on your current PSA configuration, there are three possible ways to select PDSets:

- Option A – If you have a single Priority Scheduler configuration that is used at all times, then select the PD Set name using the Select PDSet(s) dialog.
- Option B – If you have different PSF configurations in effect at different times of the day or for different days of the week or month AND these are scheduled using PSA, then define the time periods and associate the time period with the appropriate PD Sets using the Select PDSet(s) dialog.
- Option C – If you have different PSF configurations in effect at different times of the day or for different days of the week or month AND these are scheduled using some other scheduler (such as UNIX cron), then select PD Sets using the Select PDSet(s) dialog and associate each PD Set with a different time period.

The steps involved are listed below. The following steps are all sub-steps of Step 2A.

- 2A - 1. Start the Teradata Workload Analyzer
- 2A - 2. Choose *Analysis* → *Convert PDSets to Workloads*.

Click on the **DBQL Inputs** button to display the **Data Collection Interval** dialog. In this dialog, enter the date range for which you collected data.

Choose your PD Set(s) based on options A, B, or C (above) and follow the appropriate steps.

- 2A - 3A. For option A (single PS setting used at all times), do the following:
 1. Find the PD Set you wish to convert.
 2. Click on the middle column and select **Default** from the list of periods.
 3. Click on the left column to highlight the row.
 4. Click the **Proceed** button.
- 2A - 3B. For option B (multiple PD Sets scheduled at different times using PSA), do the following:
 1. Define the time periods for which the PD Sets will apply.
 2. Associate each of the time periods with a PD Set that is applicable to the time period.
 3. Click the **Proceed** button.
- 2A - 3C. For option C (multiple PD Sets scheduled at different times not using PSA), do the following:
 1. Find the PD Set you wish to convert.
 2. Click on the left column to highlight each PD Set that you wish to convert.
 3. Click the **Proceed** button.



Step 2A – Select PD Set(s) and DBQL

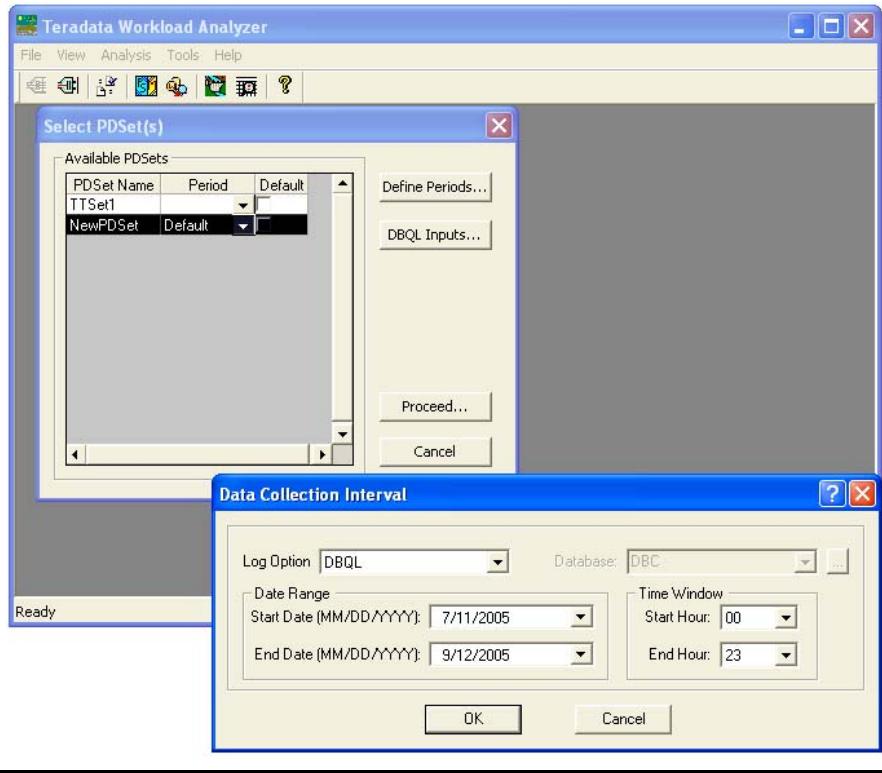
Use menu:

Choose Analysis →
Convert PDSets to
Workloads.

Use the DBQL Inputs
button to enter the date
range for collected data in
the Data Collection
Interval dialog.

For a PS setting used all
of the time.

1. Select the PD Set you
wish to convert.
2. Select default from the
list of periods.
3. Highlight the row
containing the PS set.
4. Click the Proceed
button.



Step 2B – Select Performance Groups

At this point, we have selected the PD Sets that we wish to convert and associated a period with each PD Set. We have also specified a period of time that Workload Analyzer will scan the DBQL logs for activity with the performance group names.

The Workload Analyzer (WA) searches for account names that contain the performance group name in the DBQL tables. If you chose not to use DBQL, then WA will search in the specified system catalog tables. If no account names are found, the performance group is marked inactive and no workload(s) will be created for it. However, you can force the performance group to be converted by marking it active and manually entering one or more account names.

After clicking on the Proceed button, Workload Analyzer will perform its scan and display the **Existing PSA Information** grid as shown in the example. Review the **PG Status** column. A checkmark indicates that activity for this Priority Group was seen in the DBQL log during the time period that you selected. Only those PG's with a checkmark will be converted to workloads. If you see unchecked PG's that you want to convert, add a checkmark. The classification rule for the PG will automatically include only the priority portion of the PG name.

Click on the **Convert** button.

Note: The AG Weights button displays a window showing the Existing PSF weights. This may be simply used for information and/or verification purposes.



Step 2B – Select Performance Groups

After choosing to Proceed, Workload Analyzer generates a grid of accounts using the performance group names.

Performance groups not used are marked as "inactive" and may optionally be included.

To convert the Performance Groups into Workloads, click on the Convert button.

The screenshot shows the 'Teradata Workload Analyzer - [PSA to DWM Conversion]' window. The main area displays a grid titled 'Existing PSA Setting' under 'Existing PSA Information'. The grid has columns: RP Name, PG Name, PG Status, Account, Milestone, Performance Period1, and Performance Period2. The data is organized into two sections: 'Default' and 'Tactical'. In the 'Default' section, there are four rows: L (Active), M (Active), H (Active), and R (Inactive). The 'H' row has a note '\$M_9038'. In the 'Tactical' section, there are five rows: T_L1 (Active), T_M1 (Active), T_H1 (Active), and T_R1 (Inactive). The 'T_H1' row has a note '\$T_H1\$'. The 'Performance Period1' and 'Performance Period2' columns show values like AG1, AG2, AG3, AG4, AG5, AG6, AG7, AG8, and 0. At the bottom of the window, there are buttons for 'AG Weights...', 'Convert', and 'Close'. The status bar at the bottom right says 'Logged on to TD61 as tdwm'.

RP Name	PG Name	PG Status	Account	Milestone	Performance Period1	Performance Period2
Default	L	<input checked="" type="checkbox"/> Active	\$L	Query	AG1	0
	M	<input checked="" type="checkbox"/> Active	\$M	Query	AG2	0
	H	<input checked="" type="checkbox"/> Active	\$H	Query	AG3	0
	R	<input type="checkbox"/> Inactive		Query	AG4	0
Tactical	T_L1	<input checked="" type="checkbox"/> Active	\$T_L1\$	Query	AG6	100
	T_M1	<input checked="" type="checkbox"/> Active	\$T_M1\$	Query	AG7	1
	T_H1	<input checked="" type="checkbox"/> Active	\$T_H1\$	Query	AG8	1
	T_R1	<input type="checkbox"/> Inactive		Query	AG8	0

Step 2C – Convert PD Sets to Workload Definitions (Remap As-is)

By clicking on the Convert button, Workload Analyzer has now created a set of workloads that represent your settings. On this screen, you need to choose how workloads are mapped back to the priority scheduler. You have two choices:

- **Remap as is.** This will give you exactly the same Resource Partitions and Allocation Groups that you had before enabling TASM.
- **Simplified Structure.** This option generates a Priority Scheduler configuration that matches current best practices as recommended by NCR. Your workloads will be mapped to the PSF configuration with approximately the same relative weights as before. The Simplified Structure has three resource partitions.

Miscellaneous notes about this display:

- This screen now has two tabs – Existing PSA Settings and Migrated TASM Settings.
- Workload Names – the name of the workload recommendation for each account or application appears in this box. The name consists of the prefix WD- followed by the account or application name.
- **WD-Default** – a default workload recommendation (named WD-Default) is automatically created and is used as a “No-Home WD”. Queries that do not match the characteristics of any other WD will run in this WD.

Note: The WD-Default definition can not be deleted.

TERADATA
Raising Intelligence

Step 2C – Convert PD Sets to WDs (Remap As-is)

Two choices are provided:

- Remap as-is
- Simplified

Note:
The **WD-Default** workload definition is used for any query not associated with a workload definition.

RP Name	RP Weight	AG Name	Enforcement Priority	AG Weight	Relative Weight	Previous Relative	Workload Name	Classify
Default	100	AG1	Normal	5	5	5	WD-D-\$L	Acct=\$L
		AG2	Normal	10	10	10	WD-D-\$M_9038	Acct=\$M_9038
		AG3	Normal	20	19	19	WD-D-\$M	Acct=\$M
Tactical	200	AG5	Normal	5	4	4	WD-D-\$H	Acct=\$H
		AG6	Normal	10	9	9	WD-D-\$T_L1\$	Acct=\$T_L1
		AG7	Normal	20	18	18	WD-D-\$T_M1\$	Acct=\$T_M1
		AG8	Normal	40	36	36	WD-D-\$T_H1\$	Acct=\$T_H1
Default	100	AG1	Normal	5	5	5	WD-Default	

Ready Logged on to TD61 as tdwm

Step 2C – Convert PD Sets to Workload Definitions (Simplified)

The second option is to choose the **Simplified Structure**. This option generates a Priority Scheduler configuration that matches current best practices as recommended by NCR. Your workloads will be mapped to the PSF configuration with approximately the same relative weights as before.

The Simplified Structure has three resource partitions:

- Default – only "system" work (e.g., console functions) runs in this partition.
- Tactical – only workloads that you designate as Tactical (using the Enforcement Priority) will run in this partition.
- Standard – all other work will run in this partition. Examples include batch jobs, DSS queries, etc.

You can optionally change the “Enforcement Priority” for any of the allocation groups. However, note that the Enforcement Priority for a Workload and an associated Allocation Group must be the same.

To generate candidate workload definitions, click on the Continue button.

TERADATA
Raising Intelligence

Step 2C – Convert PD Sets to WDs (Simplified)

Simplified Workload Resource partitions:

- Default
- Tactical
- Standard

Note:
The **WD-Default** workload definition is also created in the Simplified structure and is used for any query not associated with a workload definition.

RP Name	RP Weight	AG Name	Enforcement Priority	AG Weight	Relative Weight	Previous Relative	Workload Name	Classify
Default	25	L		5				
		M		10				
		H		20				
		R		40				
Standard	34	AG1	Normal	5	4	5	WD-D-\$L	Acct=\$L
		AG2	Normal	10	8	10	WD-D-\$M_9038	Acct=\$M_9038
		AG3	Normal	19	15	19	WD-D-\$M	Acct=\$M
		AG4	Tactical	4	3	4	WD-D-\$H	Acct=\$H
Tactical	67	AG5	Tactical	4	3	4	WD-D-\$T_L1\$	Acct=\$T_L1
		AG6	Tactical	9	7	9	WD-D-\$T_L1\$	Acct=\$T_L1
		AG7	Tactical	18	14	18	WD-D-\$T_M1\$	Acct=\$T_M1
		AG8	Tactical	36	29	36	WD-D-\$T_H1\$	Acct=\$T_H1
Standard	34	AG1	Normal	5	4	5	WD-Default	

Ready Logged on to TD61 as tdwm

Step 2D – Create Service Level Goals (optional)

Using the following menu selection, Workload Analyzer can generate recommended Service Level Goals for each of the workloads. Normally, you would set SLGs for critical workloads.

Tools → Calculate All WDs SLG ...

The SLG Graph option displays the Service Level Goal recommendations for each workload. This display can also be used to modify the SLG for that workload.

TERADATA
Raising Intelligence

Step 2D – Create Service Level Goals

Within Workload Analyzer, the **workload definitions** can be modified using the same options (tabs) as when directly creating a workload definition.

- Attributes
- Classification
- Exceptions
- Workload Periods

Optionally, use Workload Analyzer to generate SLGs, use:

Tools > Calculate All WDs SLG

For each workload, the **SLG Graph option** provides service level goal recommendations which can be applied.

The screenshot shows the Teradata Workload Analyzer interface with the title "Teradata Workload Analyzer - [SLG Recommendations - [WD-D-ST_M1S_1]]". The left pane displays a tree view of "Candidate Workloads" including "WD-D-\$L", "WD-D-\$M", "WD-D-\$M_9038", "WD-D-\$H", "WD-D-\$T_L1\$_.0", "WD-D-\$T_L1\$_.1", "WD-D-\$T_M1\$_.0", "WD-D-\$T_M1\$_.1", "Attributes", "SLG Graph", "WD-D-\$T_H1\$_.0", "WD-D-\$T_H1\$_.1", and "WD-Default". The main pane shows "SLG Recommendations" with "SLG Parameters" set to "Response Time" and "Workload Periods" set to "Default". A bar chart titled "Cumulative Query Count" plots the number of queries against response time (Min 0.01 sec to Max 7.34 sec). Below the chart is a table of "SLG Recommendations" with the following data:

SLG Parameters	Recommended SLG
Arrival rate (per hour)	73
Throughput (per hour)	73
Response Time (seconds)	1
Service Percent	80

Buttons for "Set Manually", "Recommend on current", "Factor By", and "Apply" are also visible.

Teradata Manager Enhancements

Teradata Manager has new features which allow the administrator to monitor workloads. These enhancements are part of the Dashboard and Trend Analysis displays.



Teradata Manager 7.1 Enhancements for TASM

Dashboard has two new tabs:

- **Workload Snapshot** – provides a single-page summary of the current workload activity (last 1 minute)
- **Workload History** – provides a single-page summary of workload activity (last hour)

New Trend Analysis charts that are available:

- Workload Trends can be graphed for the following:
 - Arrival rates
 - Rejected Query Count
 - Response times
 - Delay Query Count
- New Query Lists that are available from this display include:
 - Rejected queries
 - Delayed Queries
 - Queries that meet an exception criteria
 - Queries that exceed a Service Level Goal (SLG)

Teradata Manager – Dashboard Enhancements

The Dashboard has two new tabs:

- **Workload Snapshot** – provides a single-page summary of the current workload activity (last 1 minute)
- **Workload History** – provides a single-page summary of workload activity (last hour)

The Workload Dashboard allows you to:

- Understand how resources are being allocated to each WD
- Understand if service level guidelines are being met by each WD
- Monitor delay queue

The Workload Dashboard also allows you to see Real-Time trends in:

- Workload aggregate response time
- Workload aggregate arrival rate
- Workload delay queue length
- Workload active query count

The Sessions Report has also been updated. The sessions report has a new column which includes the workload name for each active session.

TERADATA
Raising Intelligence

Teradata Manager Dashboard

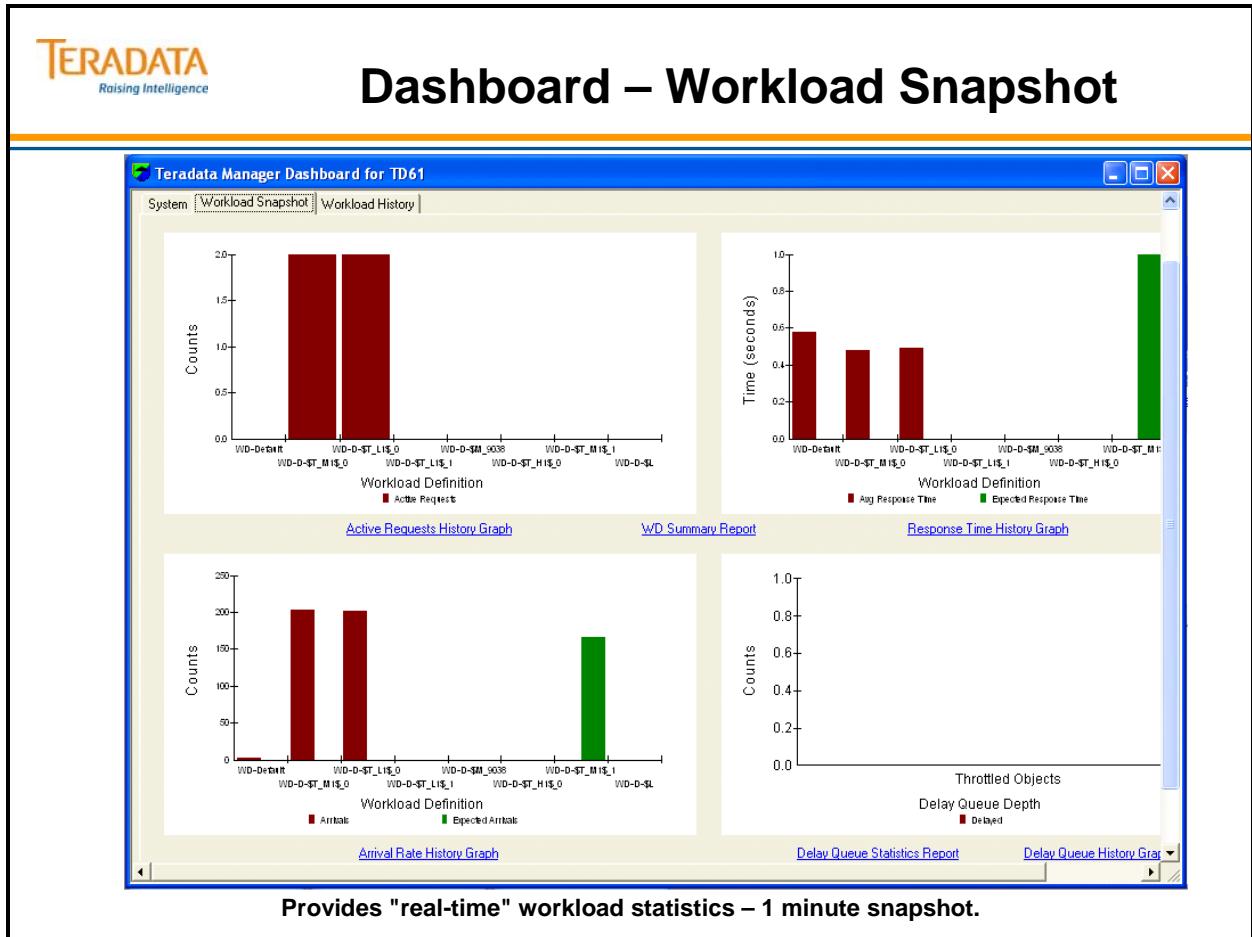
Workload Snapshot and Workload History tabs are new.

The dashboard displays the following information:

- Top Navigation:** System, Workload Snapshot, Workload History (highlighted by blue arrows).
- Status Indicators:** PE (green), AMP (green), Bynet (green), Node (green).
- Donut Charts:**
 - AMP PE Parallelism
 - Node Bynet Disk Parallelism
 - Total Blocked Active
- Line Graphs:**
 - Virtual Utilization
 - Physical Utilization
 - Sessions
- Metrics:** Transaction Rate, Response Time.

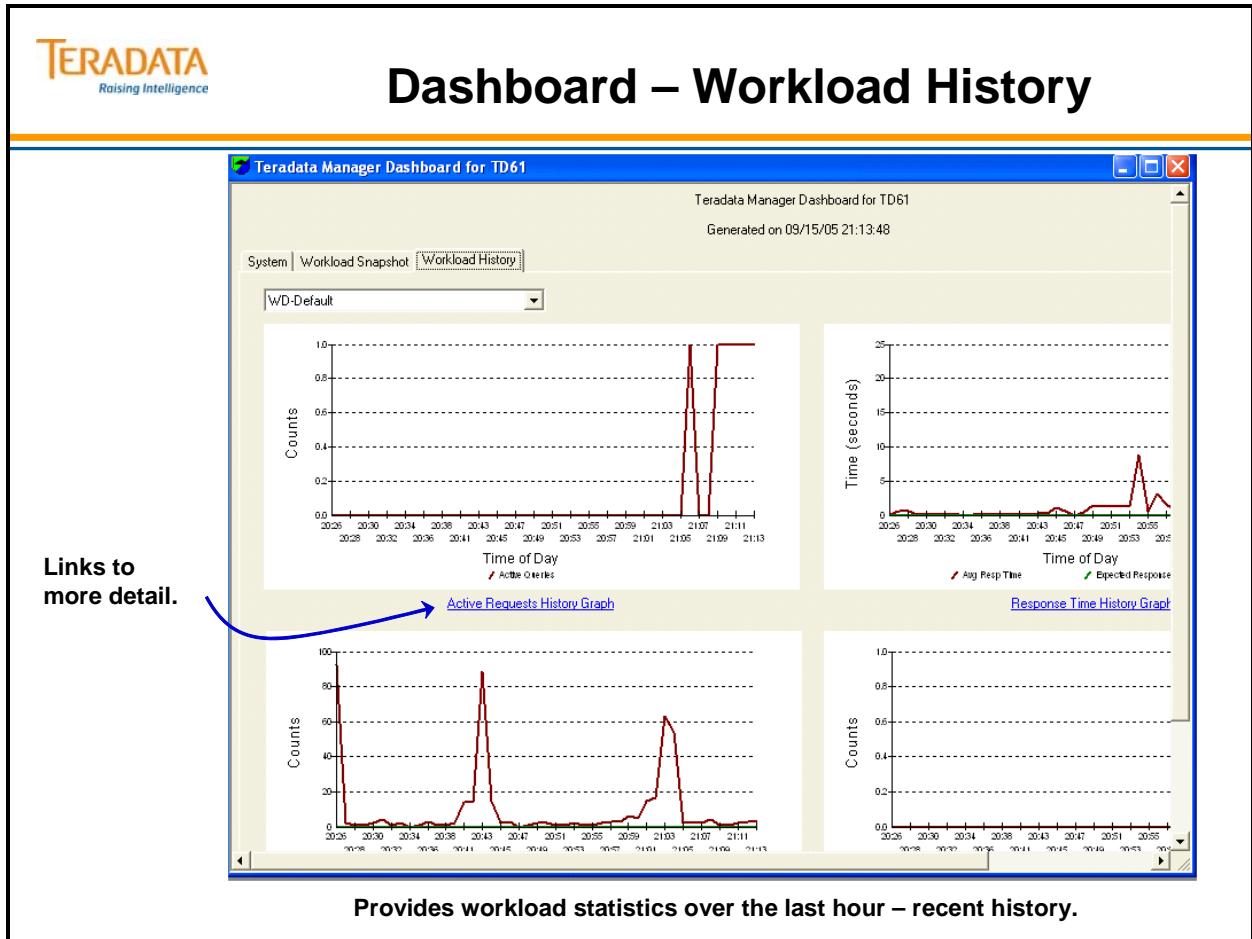
Dashboard – Workload Snapshot

The Teradata Manager Dashboard – Workload Snapshot display reports real-time statistics by workload. This display provides a single-page summary of the current workload activity. This is effectively a 1-minute snapshot of workload activity.



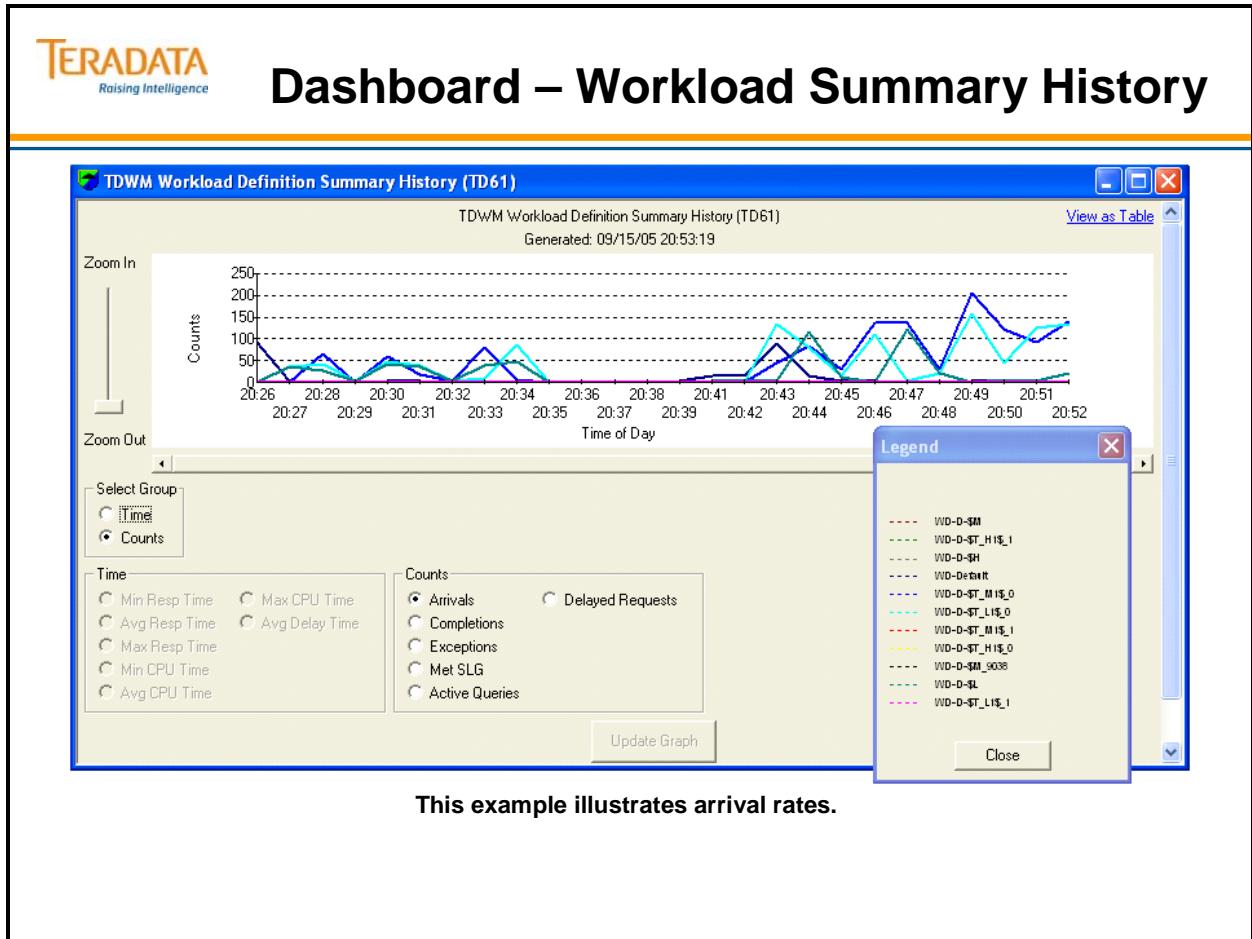
Dashboard – Workload History

The Teradata Manager Dashboard – Workload History display reports workload statistics over the last hour (recent history). This may help to visualize short term trends.



Dashboard – Workload Summary History

Both the Workload Snapshot and Workload History screens provide numerous links to more detailed displays and graphs. One example is included for arrival rates.



Teradata Manager – Trend Analysis

The Teradata Manager application has several new workload analysis displays. Workload Analysis provides the DBA with a historical view of how the system is being utilized.

To access the Workload Definition Usage Filter screen, use the following Teradata Manager menu item.

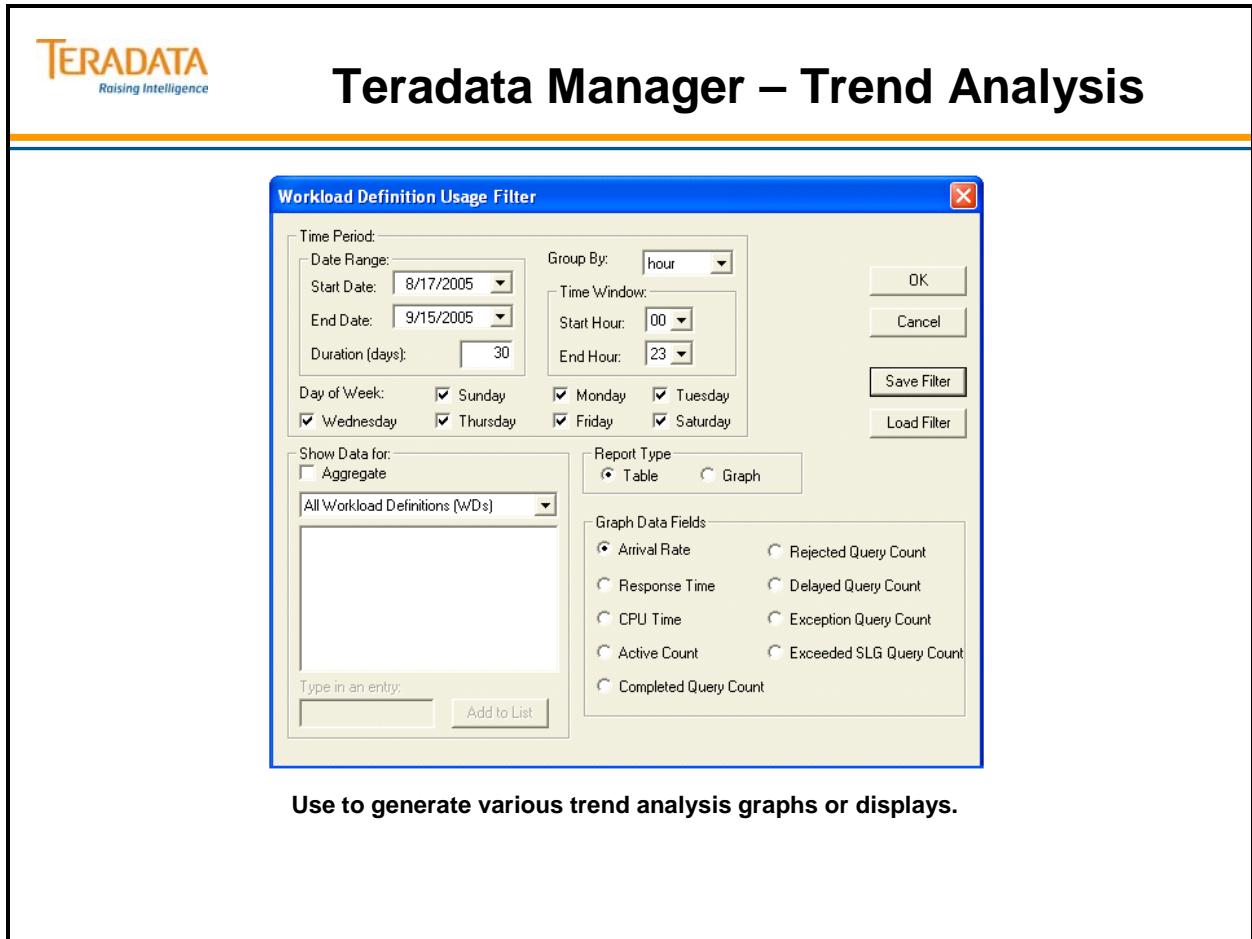
Analyze → Trends → Workload Definition Usage

With Teradata Manager 7.1, the new Trend Analysis charts that are available from this display include:

- Workload Trends can be graphed for the following:
 - Arrival rates
 - Rejected Query Count
 - Response times
 - Delay Query Count
 - etc. (as shown in the screen display)

Additional analysis displays (that have been available in Teradata Manager 7.0) that may also be used to analyze performance include:

- Resource History (RESUSAGE) Trends
- DBQL Trends



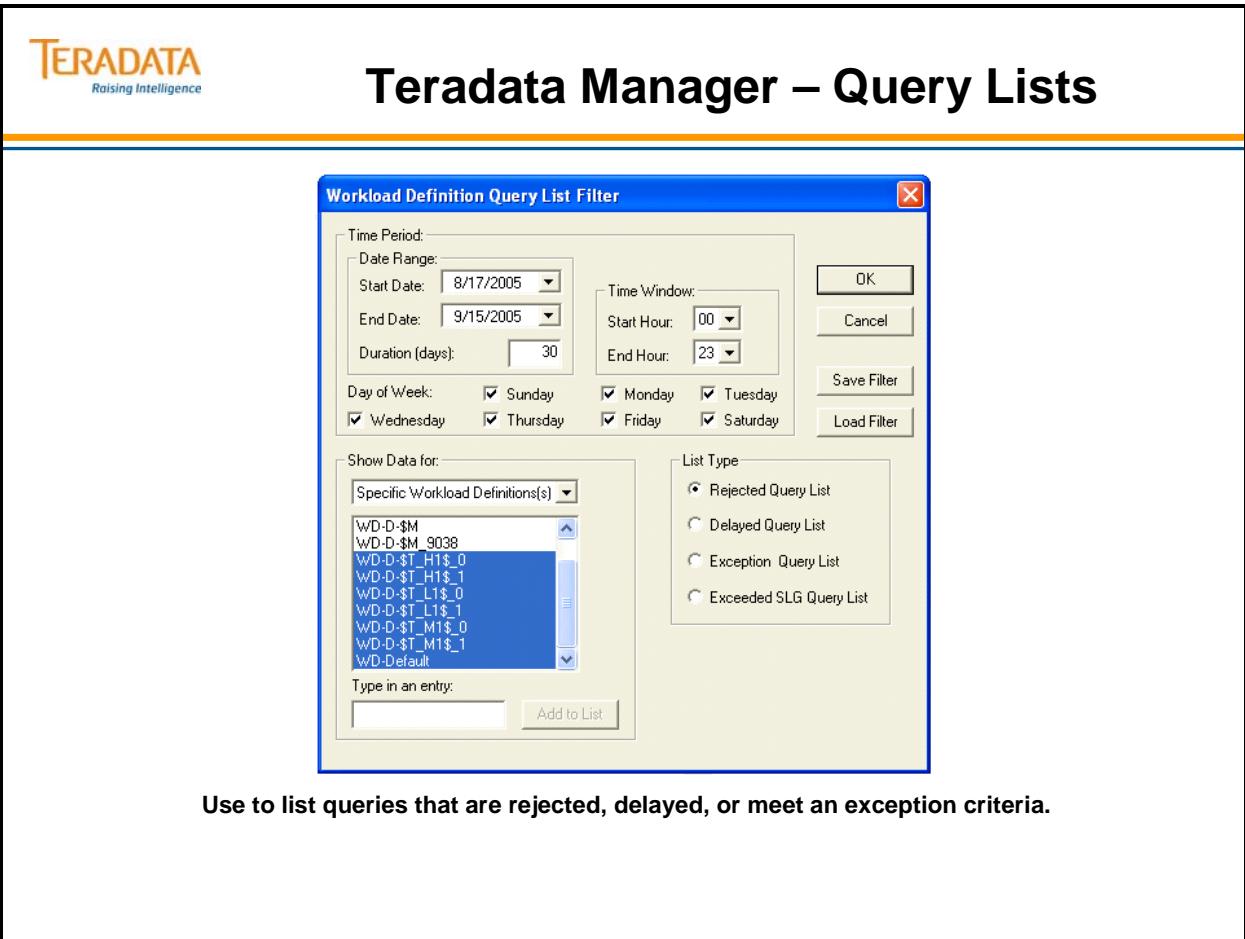
Teradata Manager – Trend Analysis – Query Lists

To access the Workload Definition Query List screen, use the following Teradata Manager menu item:

Analyze → Trends → Workload Definition Query List

With Teradata Manager 7.1, the new Query Lists that are available from this display include:

- Query Lists can be generated for the following:
 - Rejected queries
 - Delayed Queries
 - Queries that meet an exception criteria
 - Queries that exceed a Service Level Goal (SLG)



Recommendations on Features to Use

The facing page summarizes some best practice recommendations.



Recommendations on Features to Use

- If you are running out of AMP worker tasks, then ...
 - Use workload throttles to limit the number of concurrent queries.
 - Use TDWM filters to reject queries that should not run.
 - Use exceptions to abort queries that run too long.
 - Don't use “penalty box” workloads with CPU restrictions
- If the system is overloaded, then ...
 - Use workload throttles to limit the number of concurrent queries.
 - Use TDWM filters to reject queries that should not run.
 - Use exceptions to demote long-running queries to lower priority workloads
- If the tactical query response times are poor, then ...
 - Use workload throttles to limit the number of concurrent non-tactical queries.
 - Run tactical queries in their own partition with a large difference in relative weight (4x – 8x).
 - Use TDWM filters to reject queries that should not run.
 - Use exceptions to identify poorly tuned queries that do not meet Service Level Goals.

Teradata Training

Notes