**TERADATA**

**Teradata is a Relational Database Management System (RDBMS).**

Designed to run the world’s largest commercial databases.

* + Preferred solution for enterprise data warehousing.
  + Executes on UNIX MP-RAS and Windows 2000 operating systems

Compliant with ANSI industry standards.

**1**. The **Parsing Engine** dispatches request to insert a row.

**2.** The **Message Passing Layer** insures that a row gets to the appropriate AMP (Access Module Processor).

**3**. The **AMP** stores the row on its associated (logical) disk.

**4.** An AMP manages **a logical or virtual disk** which is mapped to multiple physical disks in a disk array.

**5. Linear Growth and Expandability**:

**6.** **Teradata Objects :-**

There are eight types of objects which may be found in a Teradata database/user.

**Tables** – rows and columns of data

**Views** – predefined subsets of existing tables

**Macros** – predefined, stored SQL statements

**Triggers** – SQL statements associated with a table

**Stored Procedures** – program stored within Teradata

**Join and Hash Indexes** – separate index structures stored as objects within a database

**Permanent Journals** – table used to store before and/or after images for recovery

**7. The Data Dictionary Directory (DD/D) :-**

– is an integrated set of system tables

– contains definitions of and information about all objects in the system

– is entirely maintained by the RDBMS

– is “data about the data” or “metadata”

– is distributed across all AMPs like all tables

– may be queried by administrators or support staff

– is accessed via Teradata supplied views

**8. Structured Query Language:-**

**Data Definition Language (DDL)**

* + Defines database structures (tables, users, views, macros, triggers, etc.)

CREATE DROP ALTER

**Data Manipulation Language (DML)**

* + Manipulates rows and data values

SELECT INSERT UPDATE DELETE

**Data Control Language (DCL)**

* + Grants and revokes access rights

GRANT REVOKE

**Teradata SQL also includes Teradata Extensions to SQL**

HELP SHOW EXPLAIN CREATE MACRO

**9. EXPLAIN Facility – Teradata SQL Extension :-**

The EXPLAIN modifier in front of any SQL statement generates an English translation of the Parser’s plan.

The request is fully parsed and optimized, but not actually executed.

EXPLAIN returns:

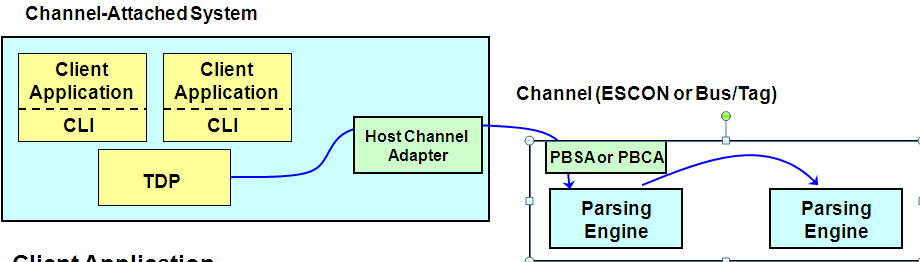
* + Text showing how a statement will be processed (a plan)
  + An estimate of how many rows will be involved
  + A relative cost of the request (in units of time)

This information is useful for:

* + predicting row counts
  + predicting performance
  + testing queries before production
  + analyzing various approaches to a problem EXPLAIN

EXPLAIN SELECT last\_name, department\_number FROM Employee**;**

**10.**

****

**Client Application**

**–** Your own application(s)  
– Teradata utilities (BTEQ, etc.)

**CLI (Call-Level Interface) Service Routines**

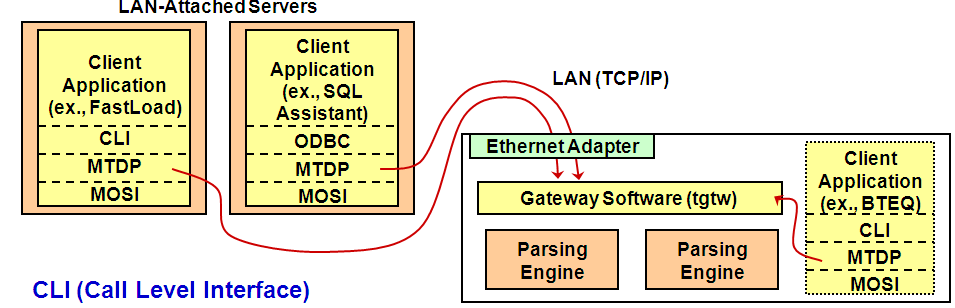
– Request and Response Control  
– Parcel creation and blocking/unblocking  
– Buffer allocation and initialization

**TDP (Teradata Director Program)**

– Session balancing across multiple PEs  
– Insures proper message routing to/from RDBMS

– Failure notification (application failure, Teradata restart)

**11. Network-Attached Client Software Overview :-**



**CLI (Call Level Interface)**

– Library of routines for blocking/unblocking requests and responses to/from the RDBMS

**ODBC™ (Open Database Connectivity) and JDBC™ (Java) Drivers**

– Use open standards-based ODBC or JDBC interfaces to provide client applications access to Teradata

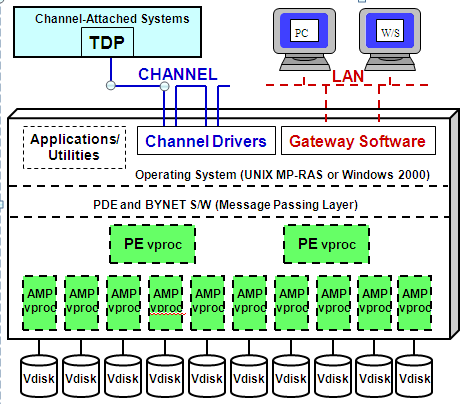
**MTDP (Micro Teradata Director Program)**

– Library of session management routines

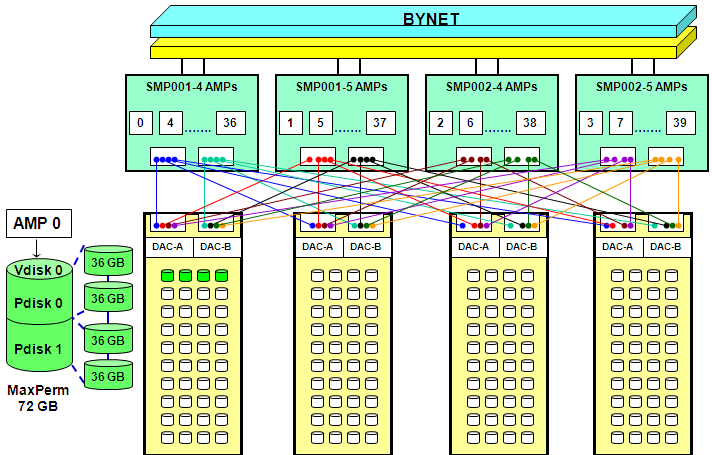
**MOSI (Micro Operating System Interface)**

– Library of routines providing OS independent interface

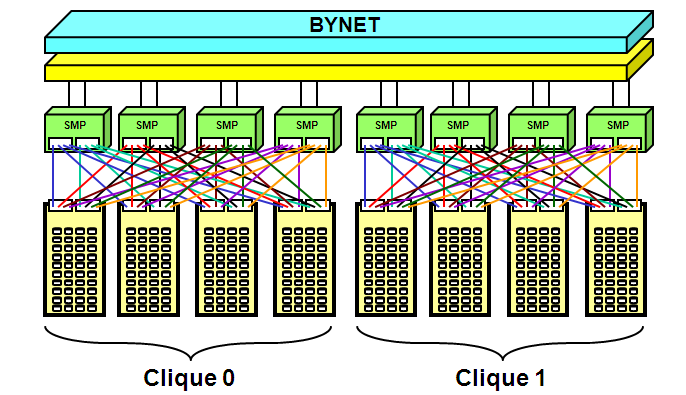
12.**Teradata Version 2 Single Node (SMP) :-**



**13. Example of 4 Node Teradata System :-**



**14. Teradata Cliques :-**



A clique is a defined set of nodes that share a common set of disk arrays.

All nodes in a clique must be able to access all Vdisks for all AMPs in the clique.

A clique provides protection from a node failure.

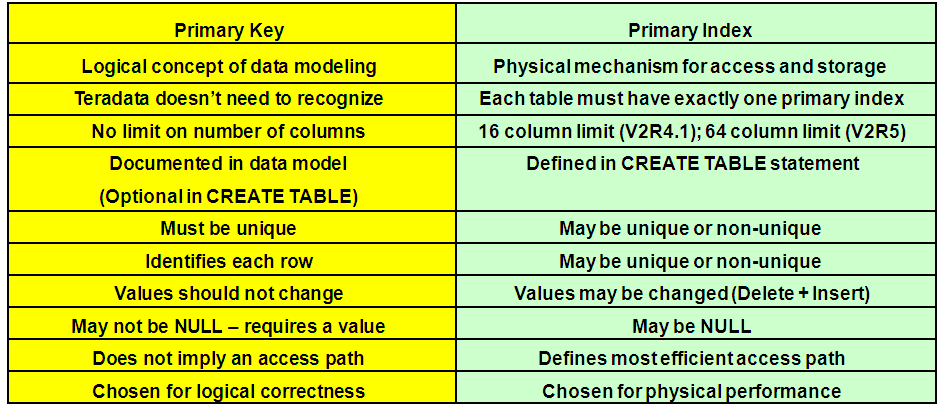
If a node fails, all vprocs will migrate to the remaining nodes in the clique (Vproc Migration).

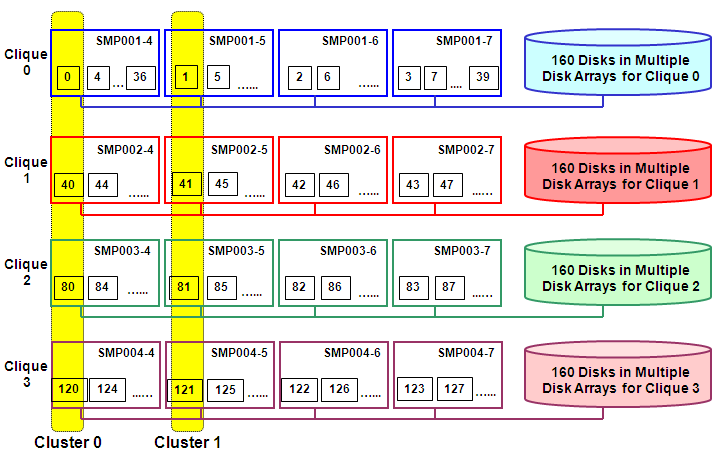
A clique can support up to 128 vprocs.

**15.Diff between User & Database:-**

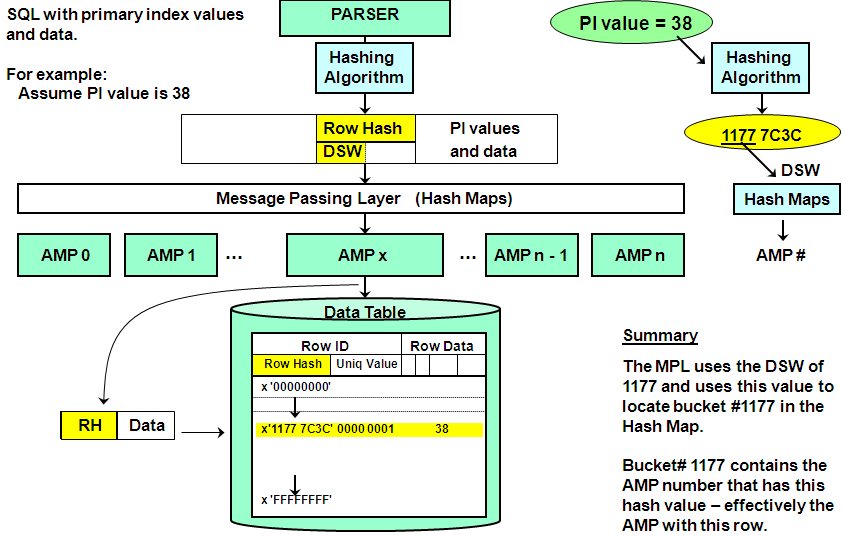


**16. Primary Keys and Primary Indexes :-**



**17. Clusters and Cliques :-**

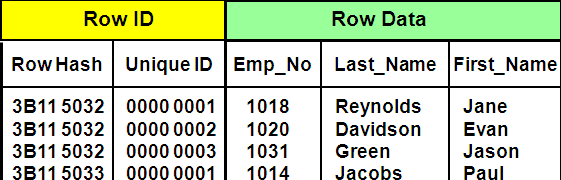
**18. Hashing Primary Index Values :-**



**19. The Row ID :-**

To uniquely identify a row, we add a 32-bit uniqueness value.

The combined row hash and uniqueness value is called a Row ID.



**20**. **Secondary Indexes:-**

A secondary Index provides an alternate path to the rows of a table.

A table can have from 0 to 32 secondary indexes.

**Secondary Indexes:**

Do not effect table distribution.

Add overhead, both in terms of disk space and maintenance.

May be added or dropped dynamically as needed.

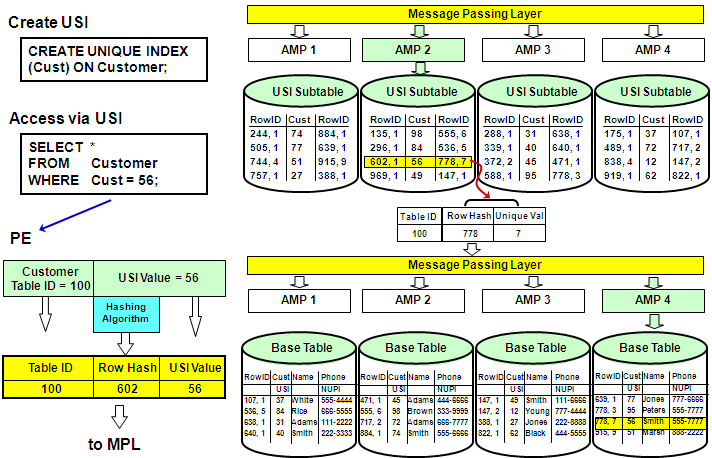
Are chosen to improve table performance.

Accessing a row via a USI is a 2 AMP operation.

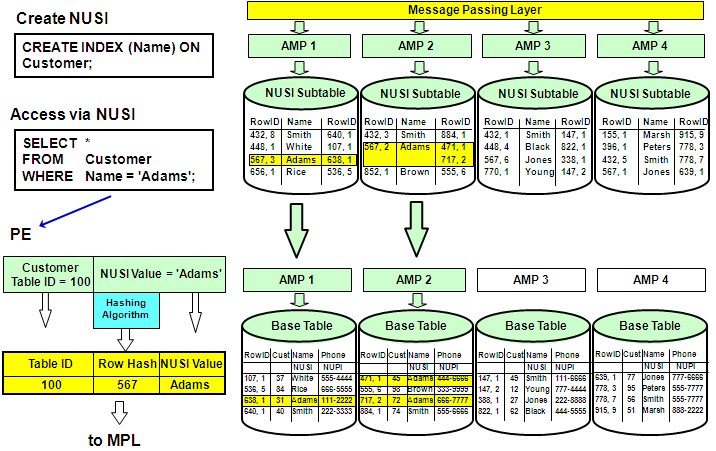
Secondary Indexes cause an internal sub-table to be built.

Dropping the index causes the sub-table to be deleted.

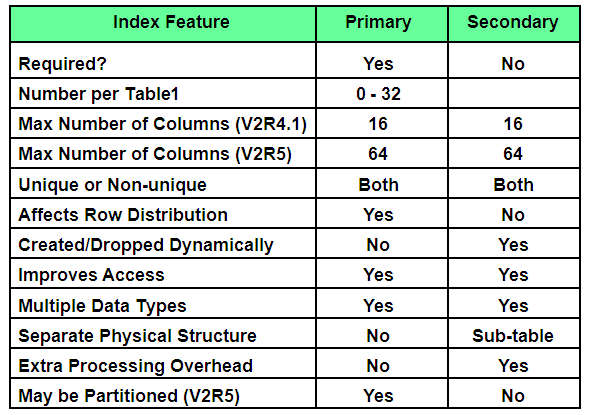
**21. Unique Secondary Index (USI) Access:-**

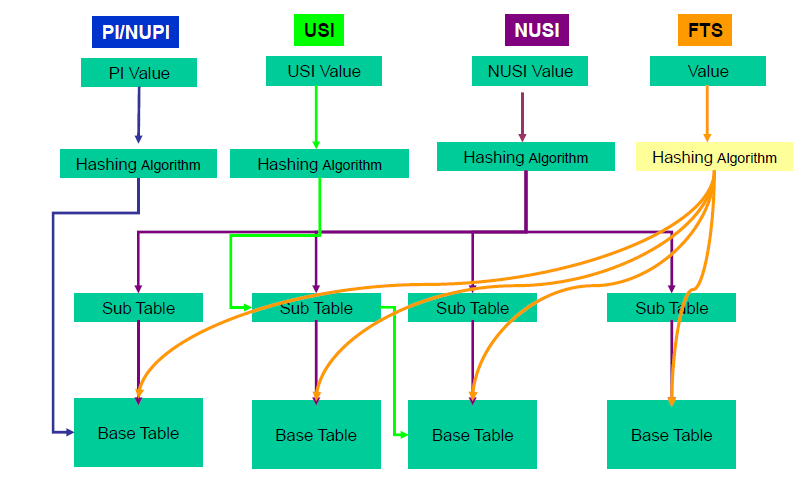


**22. Non-Unique Secondary Index (NUSI) Access :-**



**23. Comparison of Primary and Secondary Indexes :-**





**24. Fallback and RAID Protection:-**

* RAID 1 Mirroring or RAID 5 Data Parity Protection provides protection in the event of disk drive failure.
  + Provides protection at a hardware level
  + Teradata is unaware of the RAID technology used
* Fallback provides an additional level of data protection and provides access to data when an AMP is not available (not online).
* Additional types of failures that Fallback protects against include:
  + Multiple drives fail in the same drive group,
  + Disk array is not available
    - Both disk array controllers fail in a disk array
    - Two of the three power supplies fail in a disk array
  + AMP is not available (e.g., software or data error)
* The combination of RAID 1 and Fallback provides the highest level of availability.

**25. Recovery Journal for Down AMPs :-**

**Recovery Journal is:**

Automatically activated when an AMP is taken off-line.

Maintained by other AMPs in the cluster.

Totally transparent to users of the system.

**While AMP is off-line**

Journal is active.

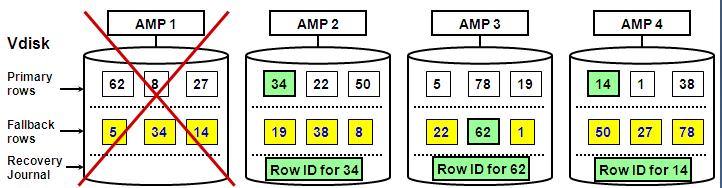
Table updates continue as normal.

Journal logs Row IDs of changed rows for down-AMP.

**When AMP is back on-line**

Restores rows on recovered AMP to current status.

Journal discarded when recovery complete.



**26. Transient Journal :-**

**Transient Journal – provides transaction integrity**

A journal of transaction “before images”.

Provides for automatic rollback in the event of TXN failure.

Is automatic and transparent.

“Before images” are reapplied to table if TXN fails.

“Before images” are discarded upon TXN completion.

**Successful TXN**

BEGIN TRANSACTION

UPDATE Row A – Before image Row A recorded (Add $100 to checking)

UPDATE Row B – Before image Row B recorded (Subtract $100 from savings)

END TRANSACTION – Discard before images

**Failed TXN**

BEGIN TRANSACTION

UPDATE Row A – Before image Row A recorded

UPDATE Row B – Before image Row B recorded

(Failure occurs)

(Rollback occurs) – Reapply before images

(Terminate TXN) – Discard before images

**27. Permanent Journal :-**

The Permanent Journal is an optional, user-specified, system-maintained journal which is used for recovery of a database to a specified point in time.

The Permanent Journal:

Is used for recovery from unexpected hardware or software disasters.

May be specified for ...

a.) One or more tables

b.) One or more databases

Permits capture of Before Images for database rollback.

Permits capture of After Images for database rollforward.

Permits archiving change images during table maintenance.

Reduces need for full table backups.

Provides a means of recovering NO FALLBACK tables.

Requires additional disk space for change images.

Requires user intervention for archive and recovery activity.

**28.** **The Stages of Database Development :-**

**29. Data Model Terminology:-**

**Business Information Model (BIM)**

-shows major entities and their relationships

-also referred to as “Business Model”

**Logical Data Model (LDM)**

-should be in Third Normal Form (3NF)

-**BIM plus**

--all tables (e.g., relationship tables)

--minor entities (e.g., code/status tables)

--Primary Key - Foreign Key relationships

--constraints

--attributes (columns)

**Extended Logical Data Model (ELDM)**

LDM plus

--demographics

--frequencies

**Physical Data Model (PDM)**

**ELDM plus**

--index selections

--any denormalizations

**30. Normalization:-**

Normalization is a technique for placing non-key attributes in tables in order to:

- Minimize redundancy

- Provide optimum flexibility

- Eliminate update anomalies

**First Normal Form (1NF**)

Attributes must not repeat within a table. No repeating groups.

**Second Normal Form (2NF)**

An attribute must relate to the entire Primary Key, not just a portion.

Tables with a single column Primary Key (entities) are always in Second Normal form.

**Third Normal Form (3NF)**

Attributes must relate to the Primary Key and not to each other.

Cover up the PK and any ND columns, and what is left must not describe each other.

**31. AMP File System:-**

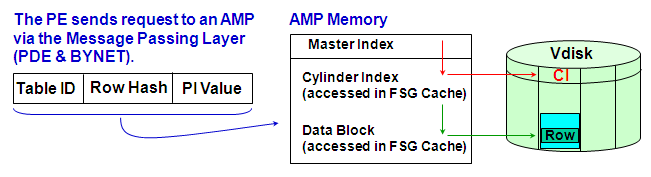
The AMP accesses its Master Index (always memory-resident).

An entry in the Master Index identifies a Cylinder # and the AMP accesses the Cylinder Index (frequently memory-resident).

An entry in the Cylinder Index identifies the Data Block.

The Data Block is the physical I/O unit and may or may not be memory resident.

A search of the Data Block locates the row(s).



**32. Space Fragmentation:-**

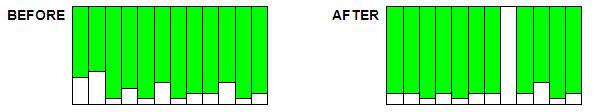
The system collects free blocks as a normal part of table maintenance.

Smaller Free Blocks become larger when adjacent blocks become free, or when defragmentation is performed on the cylinder.

**33. Cylinder Full:-**

Cylinder Full means there is no block big enough on the Free Block List.

**34. Mini-Cylpack:-**



A Mini-Cylpack moves data blocks from the data cylinder(s) to logically preceding data cylinder(s) until a single cylinder is empty.

Spool cylinders are never cylpacked.

Mini-Cylpacks indicate that the system does not have space to handle its current workload.

Excessive Cylpacks indicate too little disk space and/or spool utilization during data maintenance.

The Free Space parameter impacts how full a cylinder is filled with data loading and PackDisk

**35. Access Method Comparison:-**

**Unique Primary Index**

Very efficient

One AMP, one row

No spool file

**Non-Unique Primary Index**

Efficient if the number of rows per value is reasonable and there are no severe spikes.

One AMP, multiple rows

Spool file if needed

**Unique Secondary Index**

Very efficient

Two AMPs, one row

No spool file

**Non-Unique Secondary Index**

Efficient only if the number of rows accessed is a small percentage of the total data rows in the table.

All AMPs, multiple rows

Spool file if needed

**Full-Table Scan**

Efficient since each row is touched only once.

All AMPs, all rows

Spool file may equal the table in size

**36. COMPRESS;-**

COMPRESS all fixed length columns where at least 10% of the rows participate.

COMPRESS creates smaller rows, therefore more rows/block and fewer blocks.

**You cannot use the COMPRESS option with:**

Primary Index

Referenced and referencing data columns (PK-FK)

VARCHAR, VARBYTE, and VARGRAPHIC data types

CHAR, BYTE, and GRAPHIC data types > 255 characters

ALTER TABLE does not support compression on existing columns.

**Compression is not supported on the following data types:**

INTERVAL TIME TIMESTAMP CLOB (V2R5.1)

VARCHAR VARBYTE VARGRAPHIC BLOB (V2R5.1)

**Compression is supported on the following data types:**

DATE (4) CHAR (N) (N<256) BYTEINT (1)

SMALLINT (2) INTEGER (4) FLOAT/REAL (8)

DOUBLE (8) DECIMAL (1, 2, 4, or 8) BYTE (N) (N<256)

**Columns with frequently occurring values may be highly compressed.**

**37. Statistics:-**

Statistics basically tell the Optimizer how many rows/value there are.

The Optimizer uses statistics to plan the best way to access data.

Usually improves performance of complex queries and joins.

Stale statistics may mislead the Optimizer into poor decisions.

Helpful in accessing a column or index with uneven value distribution.

NUSI Bit Mapping is much more likely to be considered if there are collected statistics.

Statistics remain valid across a reconfiguration of the system.

COLLECT STATISTICS and DROP STATISTICS commands are DDL statements and typically are not executed during production hours.

COLLECT/DROP STATISTICS holds a row-hash Write Lock on DBC.TVFields or DBC.Indexes which prevents parsing of all new requests against the data table.

DBC.TVFields - hold statistics collected at column level

DBC.Indexes - hold statistics collected at index level

**38. EXPLAIN Facility:-**

The timings and spool sizes shown are ESTIMATES ONLY.

Spool sizes are based on dynamic sampling or statistics.

Use them as “figures of merit” for comparison purposes only.

**QUERY**

**EXPLAIN**

SELECT \* FROM daily\_sales WHERE item\_id = 5010;

EXPLANATION

------------------------------------------------------------------------------------------------------------------------

1) First, we do a single-AMP RETRIEVE step from TFACT.daily\_sales by way of the primary index "TFACT.daily\_sales.item\_id = 5010” with no residual conditions into Spool 1, which is built locally on that AMP. The size of Spool 1 is estimated with high confidence to be 731 rows. The estimated time for this step is 0.22 seconds.

-> 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.

**39. Join Processing:-**

Rows must be on the same AMP to be joined.

If necessary, the system creates spool copies of one or both rows and moves them to a common AMP.

Join processing NEVER moves or changes the original table rows.

**Typical kinds of joins are:**

Merge Join

Product Join

Nested Join

Exclusion Join

The Optimizer chooses the best join strategy based on:

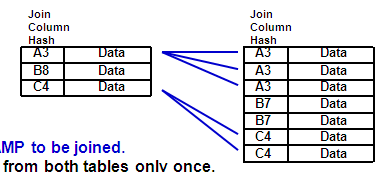
Available Indexes

Demographics (COLLECTed STATISTICS or Dynamic Sample)

EXPLAIN shows what kind of join a query uses.

**Join Redistribution**

**40. Merge Join:-**



Rows must be on the same AMP to be joined.

Merge Join reads blocks from both tables only once.

Usually chosen for an equality join condition.

Generally more efficient than a product join.

**Merge join process:**

Identify the Smaller Table.

If necessary:

Put qualifying data of one or both tables into spool(s).

Move the spool rows to AMPs based on the join column hash.

Sort the spool rows into join column hash sequence.

Compare the rows with matching join column row hash values.

Causes significantly fewer comparisons than a product join.

**41. Nested Joins:-**

This is a special join case.

This is the only join that doesn't always use all of the AMPs.

It is the most efficient in terms of system resources.

It is the best choice for OLTP applications.

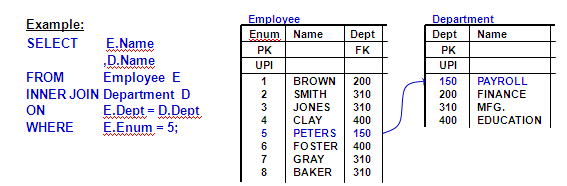
To choose a Nested Join, the Optimizer must have:

– An equality value for a unique index (UPI or USI) on Table1.

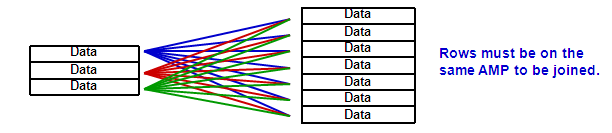
– A join on a column of that single row to any index on Table2.

The system retrieves the single row from Table1.

It hashes the join column value to access matching Table2 row(s).



**42. Product Join ;-**



Does not sort the rows.

May re-read blocks from one table if AMP memory size is exceeded.

It compares every qualifying Table1 row to every qualifying Table2 row.

Those that match the WHERE condition are saved in spool.

It is called a Product Join because:

**Total Compares = # Qualified Rows Table 1 \* # Qualified Rows Table 2**

The internal compares become very costly when there are more rows than AMP memory can hold at one time.

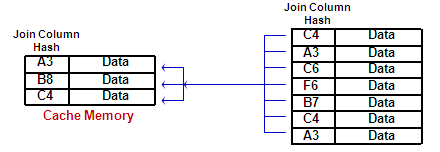
They are generally unintentional and often give meaningless output.

Product Join process:

Identify the Smaller Table and duplicate it in spool on all AMPs.

Join each spool row for Smaller Table to every row for Larger Table.

**43. Hash Join:-**



This optimizer technique effectively places the smaller table in cache memory and joins it to the larger table in unsorted spool.

**Row Hash Join Process:**

Identify the smaller table.

Redistribute or duplicate the smaller table in memory across the AMPs.

Sort the cache memory into join column row hash sequence.

Hold the rows in memory.

Use the join column row hash of the larger table to binary search memory for a match.

This join eliminates the sorting, and possible redistribution or copying, of the larger table.

EXPLAIN plans will contain terminology such as “Single Partition Hash Join”.

**44. Exclusion Joins:-**

Find rows that DON'T have a match.

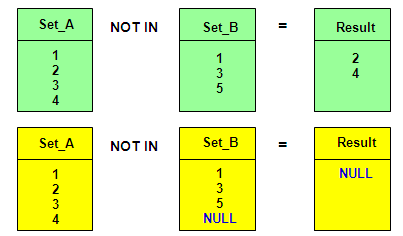
May be done as merge or product joins.

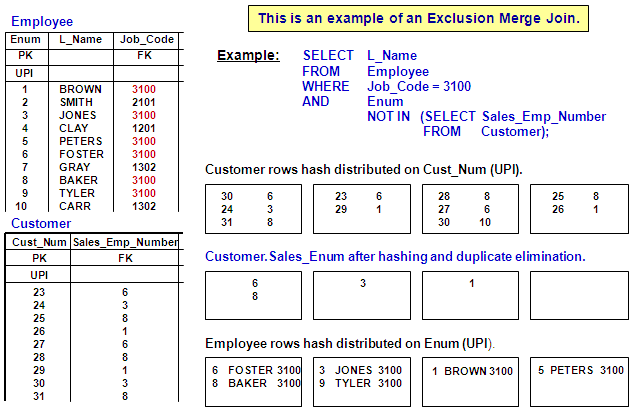
Are caused in NOT IN subqueries and EXCEPT operations.

Use 3-value logic (= , <> , unknown) on nullable columns.

Define NOT IN columns as NOT NULL on the CREATE TABLE if possible.

Use: WHERE colname IS NOT NULL in queries against nullable join columns.





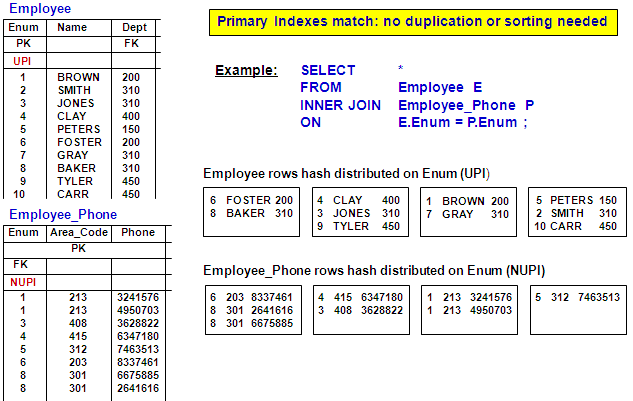
**45. Join Distribution Strategies:-**

Join costs rise with the number of rows that are moved and sorted.

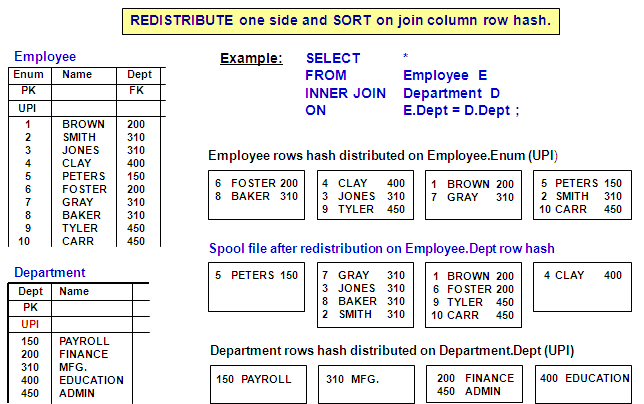
Join plans for the same tables change as the demographics change

MERGE JOIN - Rows must be in JOIN COLUMN Row Hash sequence.

1. **Do nothing if Primary Indexes match and are the join columns**

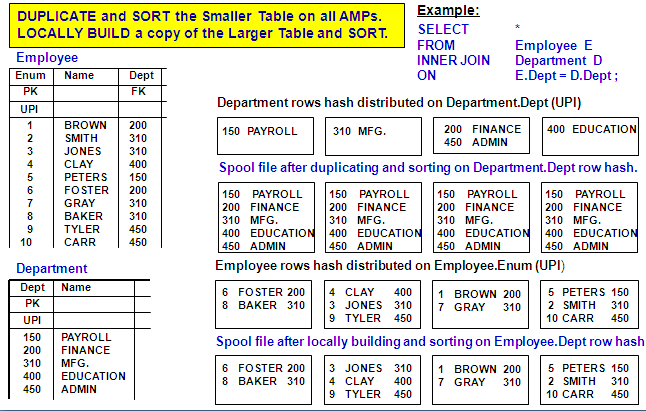


1. **REDISTRIBUTE one or both sides (depending on the Primary Indexes used in the join) and SORT on join column row hash.**

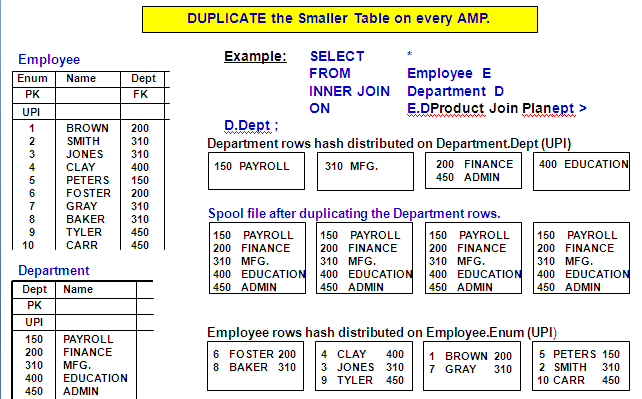


1. **DUPLICATE the smaller Table on all AMPs and SORT on join column row hash.**

**A spool copy is built LOCALLY of the Larger Table and SORT on join column row hash.**



**Product Join Plan :-**



**46. Additional Index Choices:-**

As part of the physical design process, the designer may choose to implement some of the following optional indexes for performance reasons.

**Join Indexes**

Pre-join multiple tables.

Distribute the rows of a single table on the hash value of a foreign key value.

Aggregate one or more columns of a table or tables into a summary table.

**Sparse (Join) Indexes – V2R5 feature**

You can limit the rows (WHERE clause) that are included in the join index to a subset of the rows in the table based on an SQL query result.

**Global (Join) Indexes – V2R5 feature**

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.

**Hash Indexes – V2R4.1 feature**

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

Materialized Views are implemented as Join Indexes. Join Indexes improve query performance at the expense of slower updates and increased storage.

Note: These indexes are limited to 16 columns (V2R4.1) and 64 columns (V2R5)

# 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

**Multiple 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

**Benefits of Join Indexes**:

If possible, the optimizer will use a Join Index rather than access the tables directly.

For known queries, this typically will result in much better performance.

A Single-Table Join Index can eliminate data distribution.

A Multi-Table Join Index pre-joins the data and can eliminate join processing.

An Aggregate Join Index can eliminate aggregation processing.

Provides the optimizer with additional options and the optimizer may use the join index if it “covers” the query.

Join Indexes are automatically updated as the table rows are updated.

A Join Index may not be accessed directly.

Note: 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 include:**

Take up PERM space.

Fallback protection for a Join Index is available with Teradata V2R4.

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.

A Trigger and a Join Index cannot be defined on the same table.

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

You may create non-unique secondary indexes on its columns.

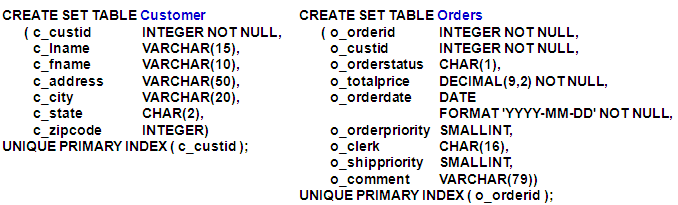
Perform COLLECT/DROP STATISTICS, DROP, HELP, and SHOW statements.

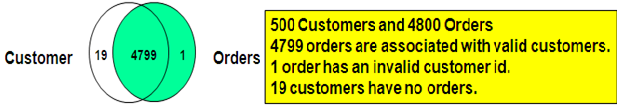
**Unlike base tables, you cannot do the following:**

Directly query or update join index rows.

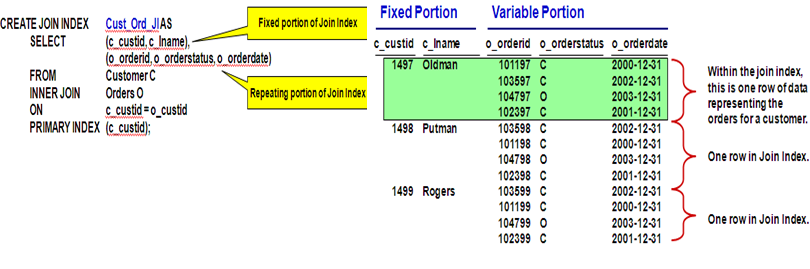
Create unique indexes on its columns.

Store and maintain arbitrary query results such as expressions.





**Creating a Join Index – Multiple Tables**

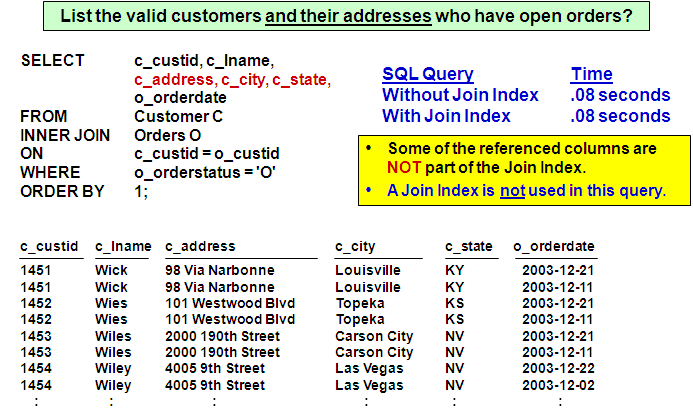


**Example 1 – Does a Join Index Help?**

List the valid customers who have open orders?



**Example 2 – Does a Join Index Help?**



**Creating a Join Index – Single Table**

CREATE JOIN INDEX Orders\_JI AS

SELECT

o\_orderid,

o\_custid,

o\_orderstatus,

o\_totalprice,

o\_orderdate

FROM Orders

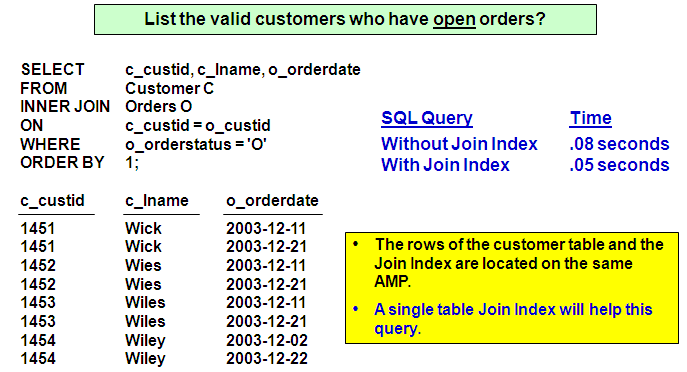
PRIMARY INDEX (o\_custid);

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?**



**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.

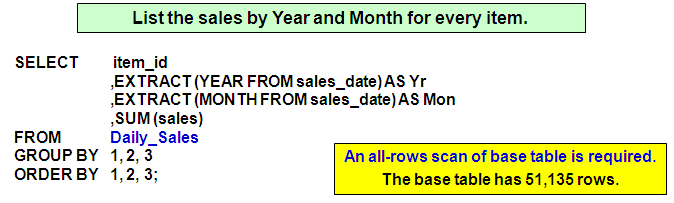
**Privileges required to create any Join Index:**

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**



**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 AS**

**SELECT**

**item\_id AS Item**

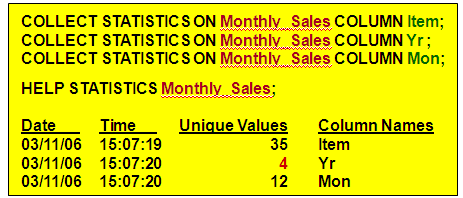
**,EXTRACT (YEAR FROM sales\_date) AS Yr**

**,EXTRACT (MONTH FROM sales\_date) AS Mon**

**,SUM (sales) AS Sum\_of\_Sales**

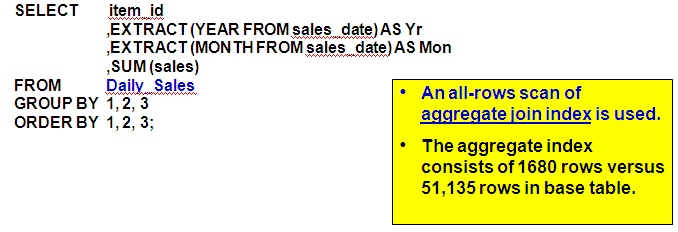
**FROM Daily\_Sales**

**GROUP BY 1, 2, 3;**



**Aggregation with an Aggregate Index**

List the sales by Year and Month for every item.



1. **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**

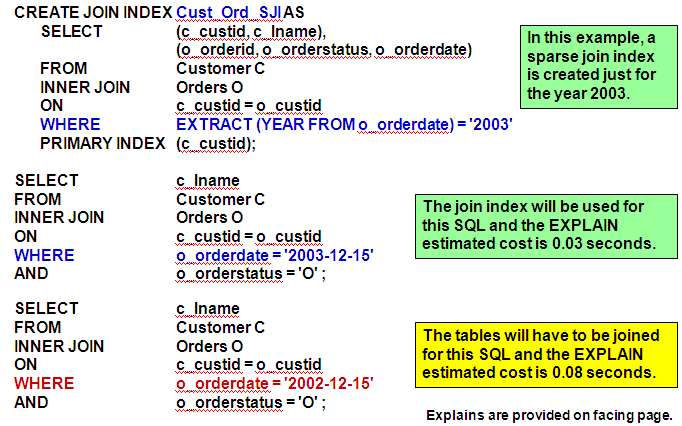
A sparse index can focus on the portion of the table(s) that are most frequently used.

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)

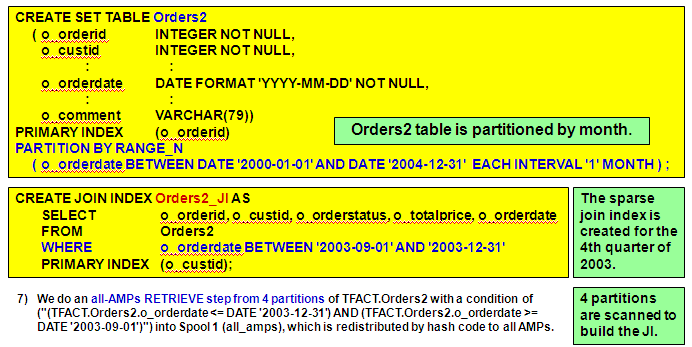


**Creating a Sparse Join Index on Partitions**

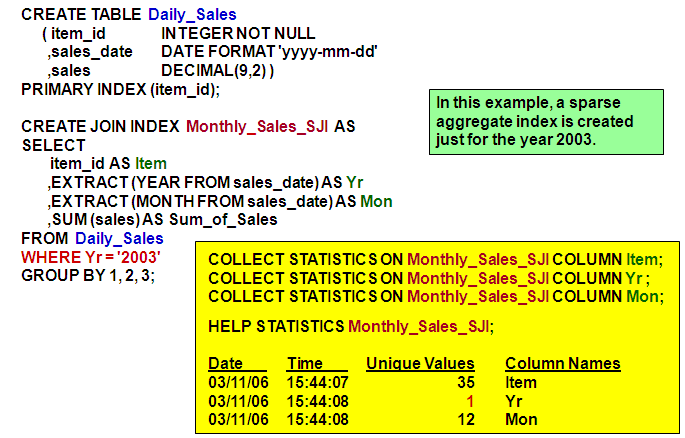
If the base table is partitioned, a sparse join index may be created for a partition or partitions – only the “partitions of interest” are scanned.

Creation time of sparse join index is faster because of partition elimination.

Partition scan of base table instead of full table scan.



**Creating a Sparse Aggregate Join Index**

****

**3. 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 50 columns. The “join index” can only include (at most) 64 columns.

An SQL request that references a column not in this set causes an actual join to occur.

You can include (as part of the 64 columns) two columns named 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.

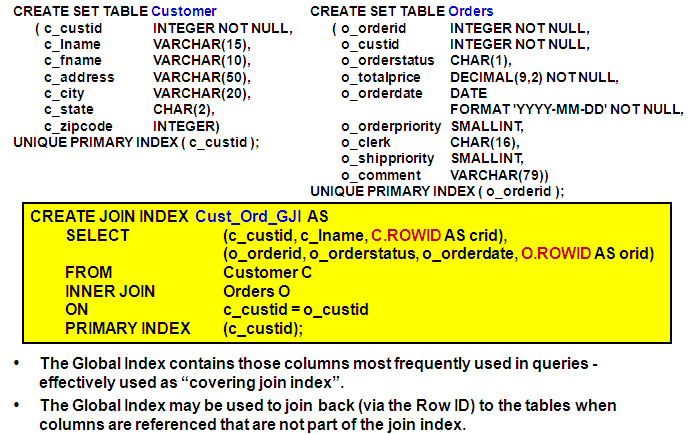
Useful when a column is used as a secondary index, but the typical number of rows for a value is much less than the number of AMPs in the system.

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**

****

**4. Hash Indexes**

Hash Indexes (V2R4.1 feature) 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 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:**

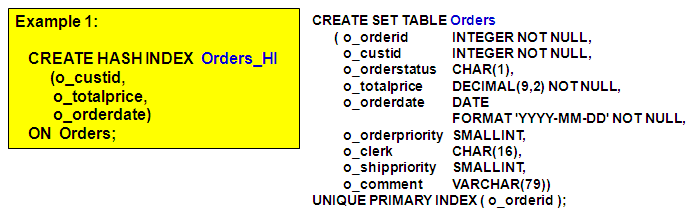
Automatically contains base table PI value as part of hash index subtable row.

No aggregation operators are permitted.

They are always defined on a single table.

No secondary indexes may be built on the hash index.

**Creating a Hash Index**



**Characteristics of this Hash Index are:**

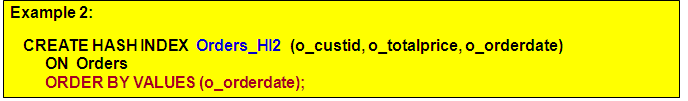
Hash index subtable rows are AMP-local to the base table rows and are in row hash sequence based on base table Primary Index value (e.g., o\_orderid).

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.

****

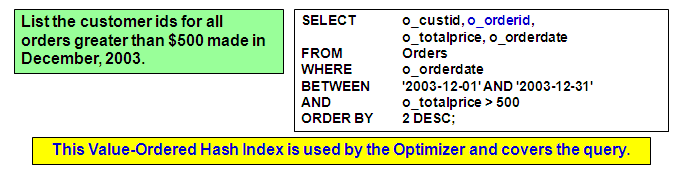
**Characteristics of this Hash Index are:**

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.

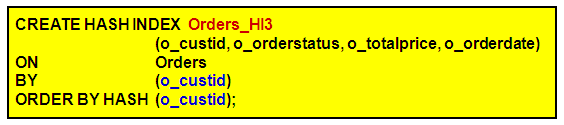
****

**A Hash Index can be ordered by value or hash.**

The “primary” index of a join index can only be ordered by hash.

Secondary indexes may be created for a join index that are 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”.

****

**Characteristics of this Hash Index are:**

Hash index subtable rows are hash distributed by “o\_custid” value.

The BY option is required when ORDER BY HASH is specified.

Hash index subtable rows include specified columns and the PI value.

This can be used as a “covering index”.

**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 a 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.

The “primary” index of a join index can only be ordered by hash.

**47.** **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**

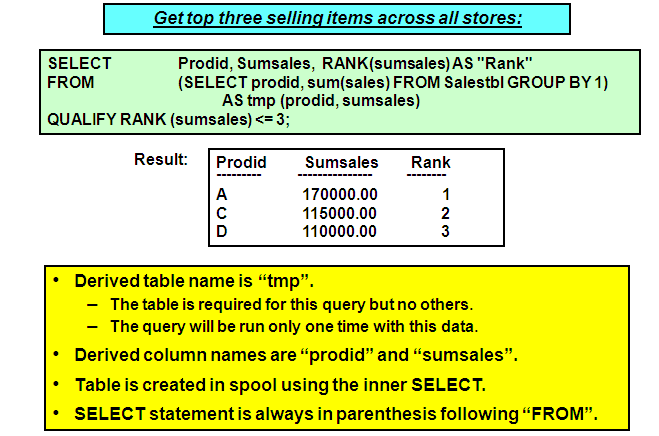
Local to a session – **uses TEMPORARY space**

Uses CREATE GLOBAL TEMPORARY TABLE syntax

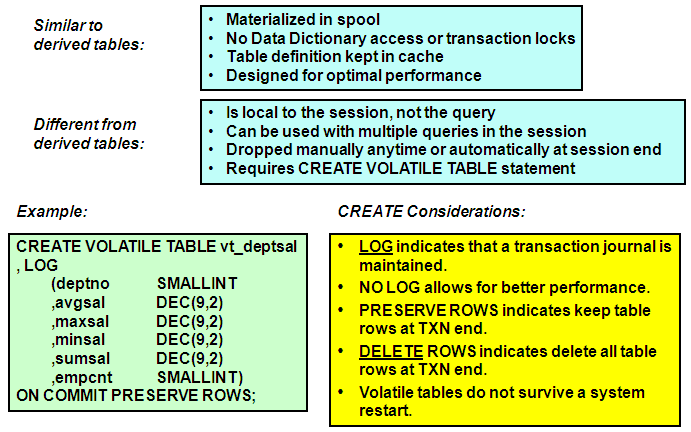
Materialized instance of table discarded at session end

Creates and keeps table definition in data dictionary

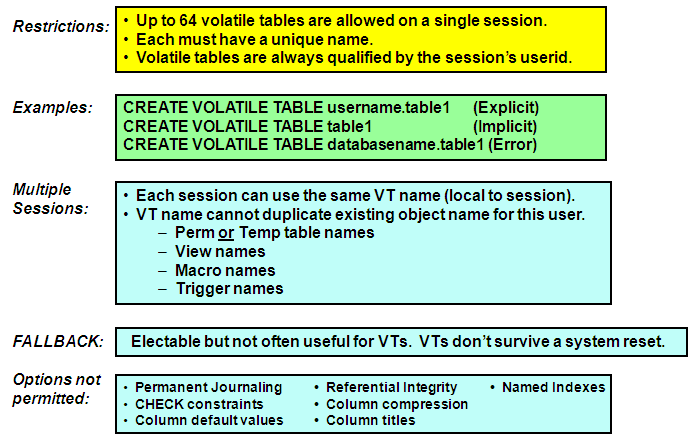
1. **Derived Tables Revisited**

****

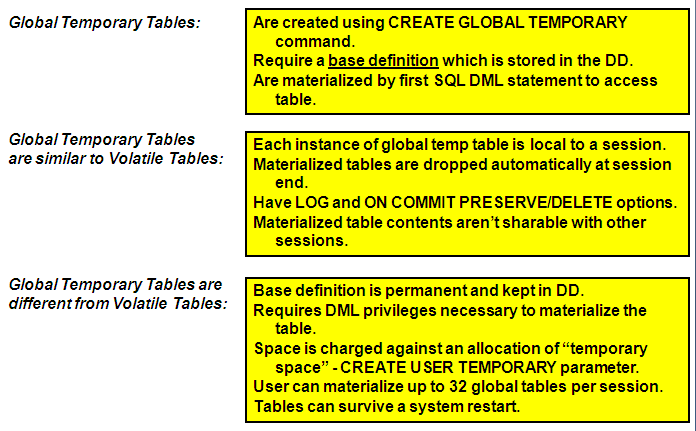
1. **Volatile Tables**

****

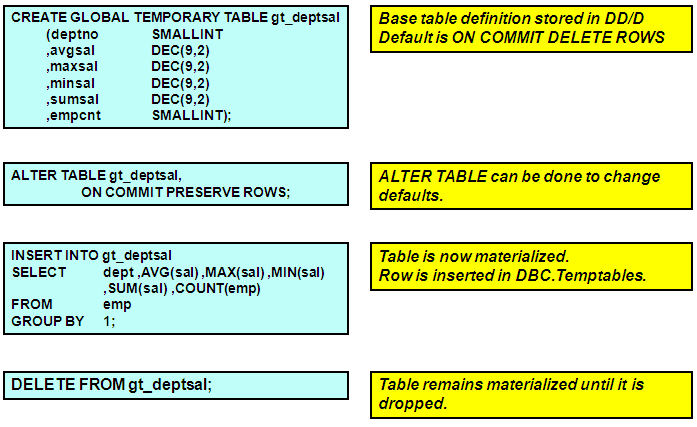
**Volatile Table Restrictions:**

****

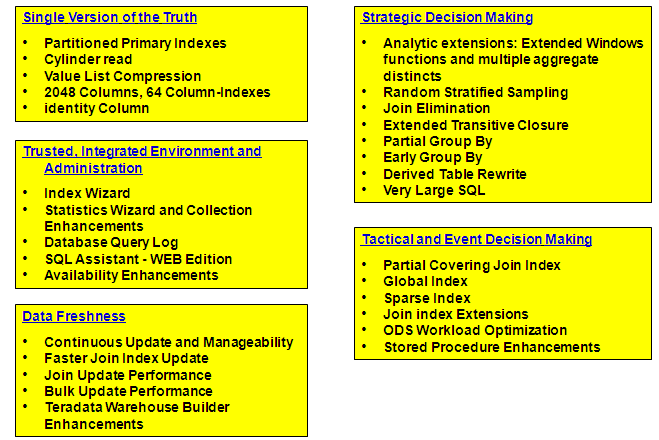
1. **Global Temporary Tables:-**

****

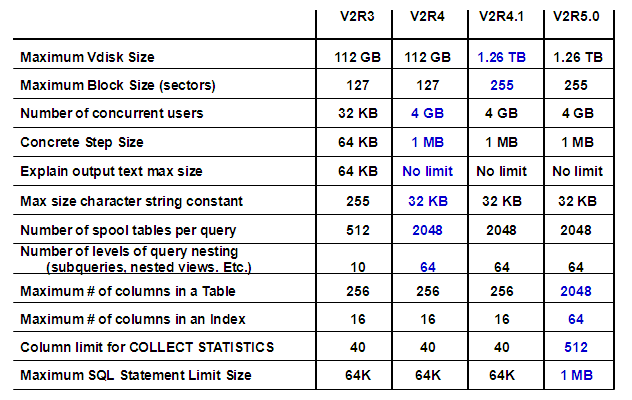
**Creating Global Temporary Tables:-**

****

**V2R5 – New Features:-**

****

**Teradata Limits (Different Releases):-**

****

**Transferring Large Amounts of Data :-**

****

**Application Utility Summary :-**

**BTEQ supports SELECT, INSERT, UPDATE, DELETE.**

**FastLoad, MultiLoad, and TPump transfer data from the host to Teradata.**

**FastExport transfers data from Teradata to the host.**

**BTEQ INSERT/SELECT and DELETE (ALL) can provide a fast effective method to perform some tasks.**

**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 Warehouse Builder (TWB) 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.**

**48.BTEQ:-**

Batch-mode utility for submitting SQL requests to the Teradata database.

Runs on every supported platform — laptop to mainframe.

Flexible and easy-to-use report writer.

Exports data to a client system from the Teradata database:

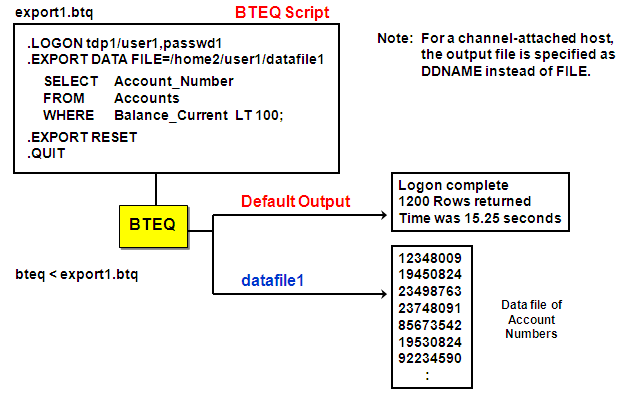
--As displayable characters suitable for reports, or

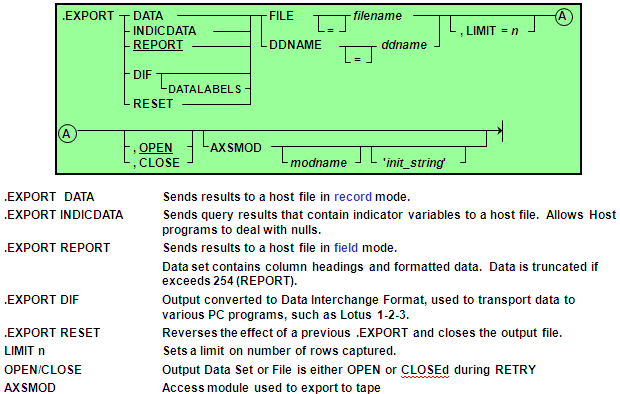
--In native host format, suitable for other applications.

Reads input data and imports it to the Teradata database as INSERTs, UPDATEs or DELETEs.

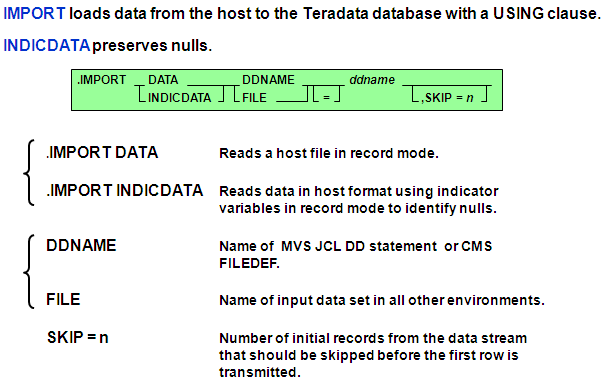
Limited ability to branch forward to a LABEL, based on a return code or an activity count.

BTEQ does error reporting, not error capture.

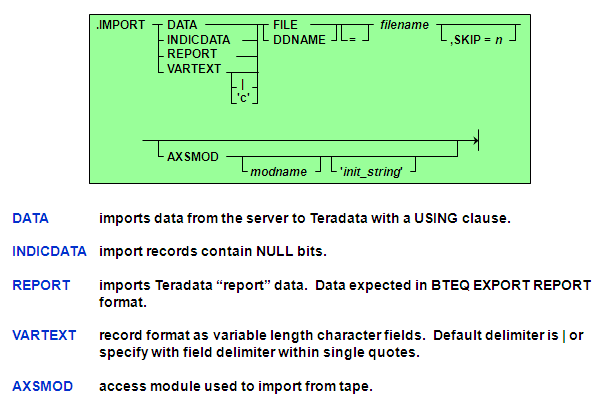




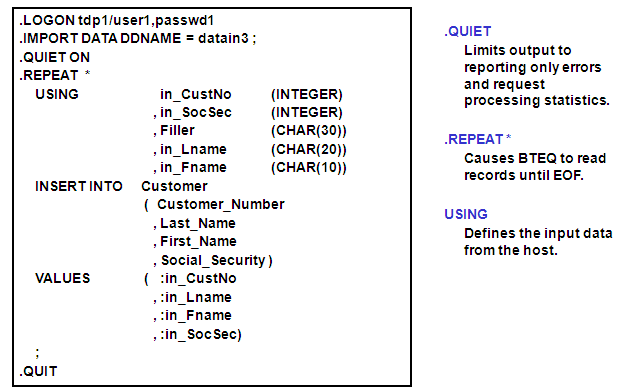
**.IMPORT  
(for Channel-Attached Systems)**

****

**.IMPORT  
(for Network-Attached Systems)**

****

**BTEQ IMPORT   
(Data Load from the Host)**

****

**49. FastLoad:-**

Fast batch mode utility for loading new tables onto the Teradata database

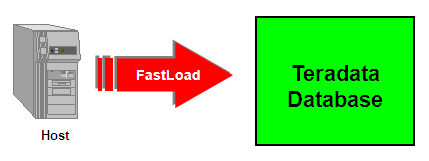
Can reload previously emptied tables

Full Restart capability

Error Limits and Error Tables, accessible using SQL

Restartable INMOD routine capability

Ability to load data in several stages



**Purpose**

Load large amounts of data into an empty table at high speed.

Execute from NCR servers, channel, or network-attached hosts.

**Concepts**

Loads into an empty table with no secondary 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 Teradata Database will accommodate up to 15 FL/ML/FE applications at one time.

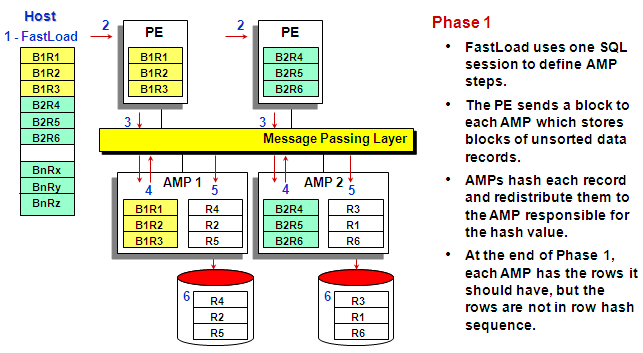
Tables defined with Referential integrity, secondary indexes, Join Indexes, Hash Indexes, or Triggers cannot be loaded with FastLoad.

--Tables with Soft Referential Integrity (V2R5) can be loaded with FastLoad.

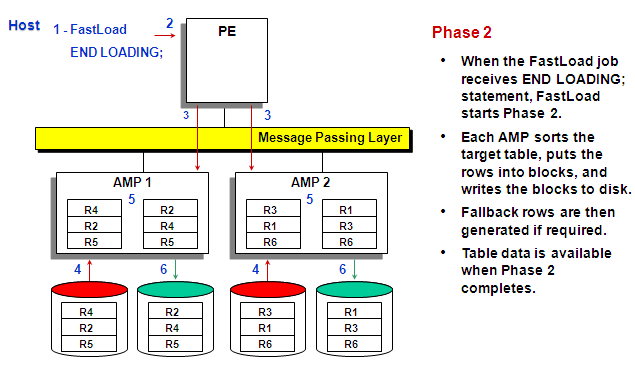
Duplicate rows cannot be loaded into a multiset table with FastLoad.

If an AMP goes down, FastLoad cannot be restarted until it is back online.

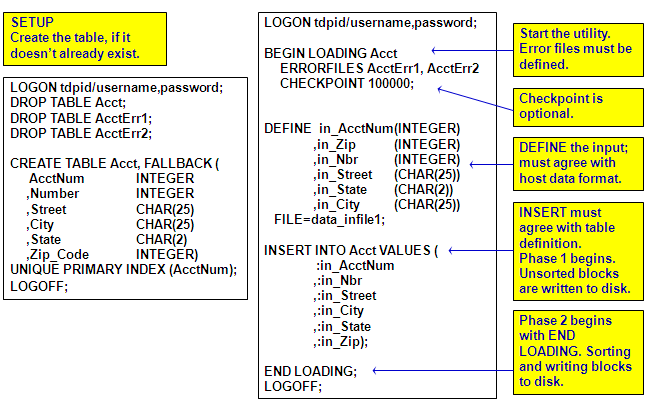
**FastLoad Phase 1**

****

**FastLoad Phase 2**

****

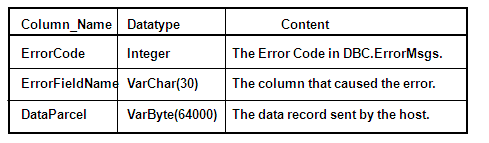
**A Sample FastLoad Script**

****

**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:

****

**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.

**Restarting FastLoad**

**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.

**FastLoad Features and Characteristics:**

* + Excellent utility for loading new tables from the host or server.
  + Loads into an empty table with no secondary indexes.
  + 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

**50. FastExport:-**

Exports large volumes of formatted data from Teradata to a host file or user-written application.

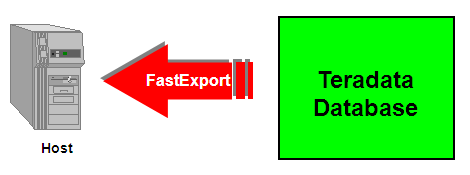
Takes advantage of multiple sessions.

Export from multiple tables.

Uses Support Environment.

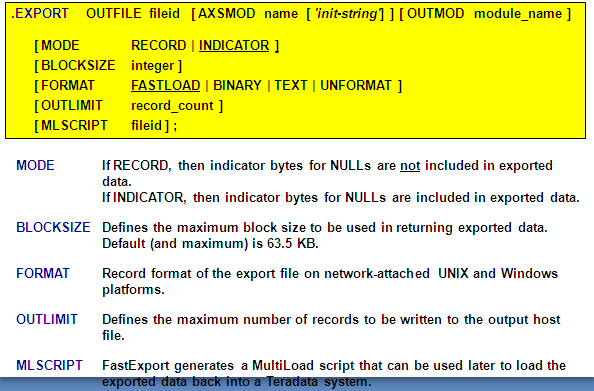
Fully automated restart.

Uses one of the “Loader” slots.

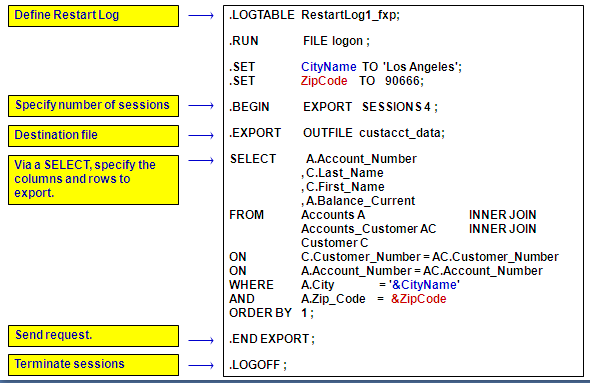


**.BEGIN and .END EXPORT**





**A FastExport Script**

****

**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.

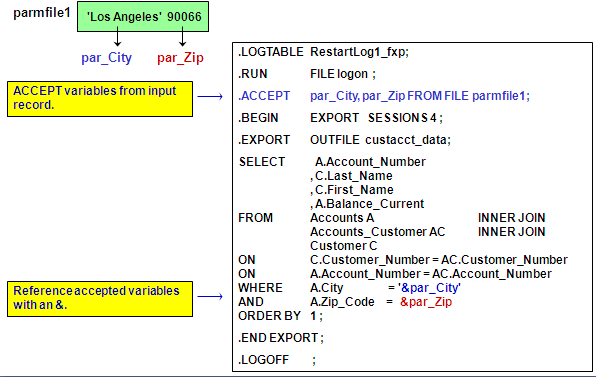
Read input variables from a host input data file described by the .LAYOUT command.

Apply each input variable value to every SELECT in the exact order listed in the FastExport script before reading the next.

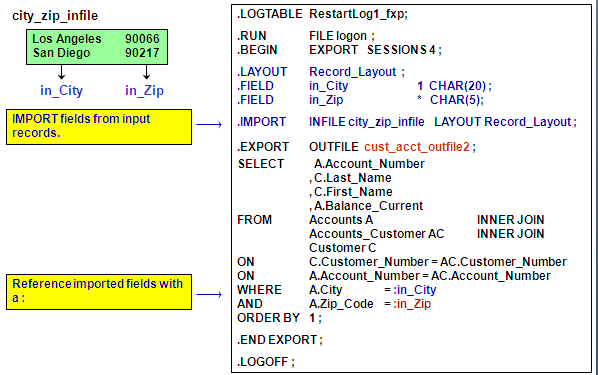
Defines a host file as the source of the data values required for the SELECT REQUEST.

Permits the use of a user-written INMOD routine to (optionally) read and (always) process the input record before passing it to the utility.

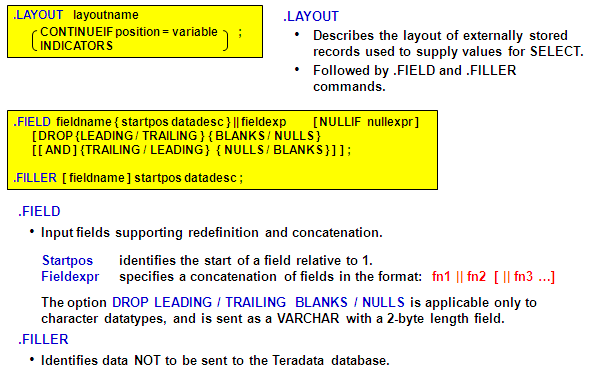
**A FastExport Script with ACCEPT**

****

**A FastExport Script with LAYOUT**

****

**.LAYOUT, .FIELD and .FILLER**

****

**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.
* Teradata accommodates not more than 15 ‘LOAD’ applications at any one time (FastLoad, MultiLoad, FastExport).

**51. 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.

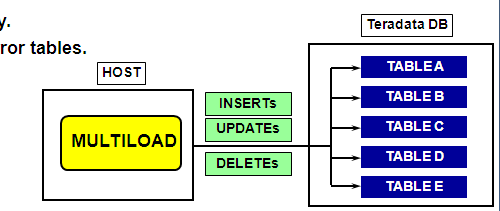
Affected data blocks only written once.

Host and LAN support.

Full Restart capability.

Error reporting via error tables.

Support for INMODs.

****

**MultiLoad Limitations**

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.

**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.

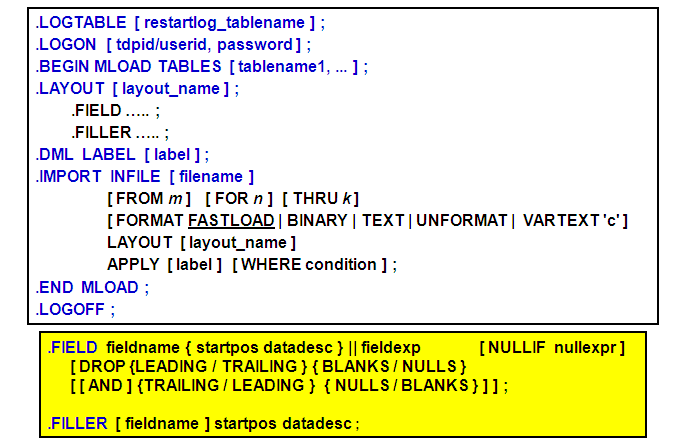
Prevents lengthy rollbacks of aborted jobs.

Allows for maximum access to table during processing.

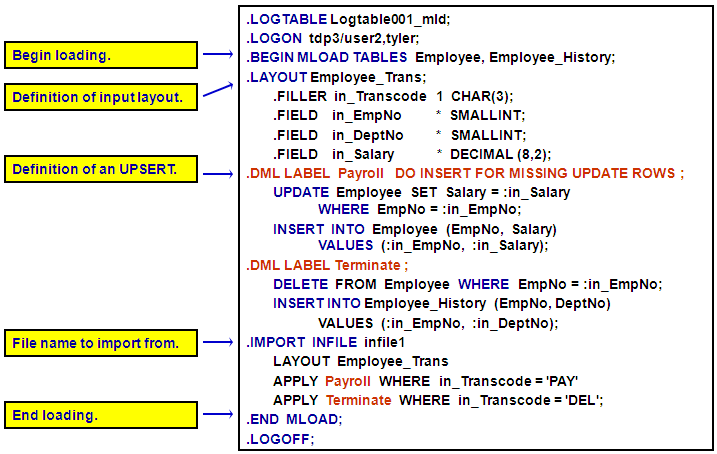
Posts errors to special error tables.

Provides extensive processing statistics.

**Basic MultiLoad Statements**

****

**Sample MultiLoad IMPORT Task**

****

**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.

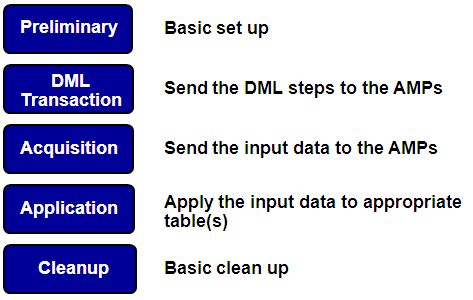
Input error limits may be specified as a number or percentage.

Allows restart and checkpoint during each operating phase.

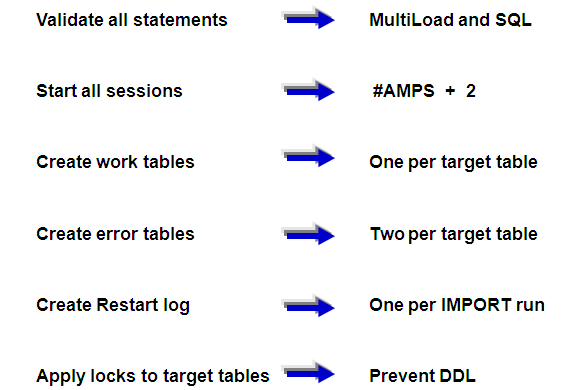
IMPORT tasks cannot be done on tables with USI’s, Referential Integrity, Join Indexes, Hash Indexes, or Triggers.

--With V2R5, IMPORT tasks can be done on tables defined with “Soft Referential Integrity”.

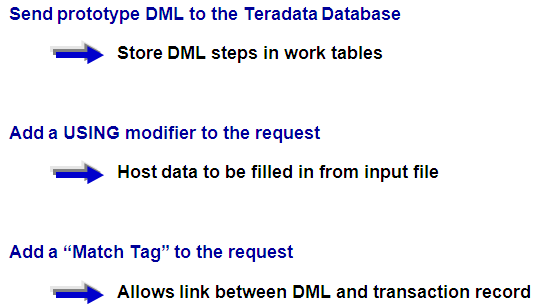
**5 Phases of IMPORT Task**



**Phase 1: Preliminary**



**Phase 2: DML Transaction**

****

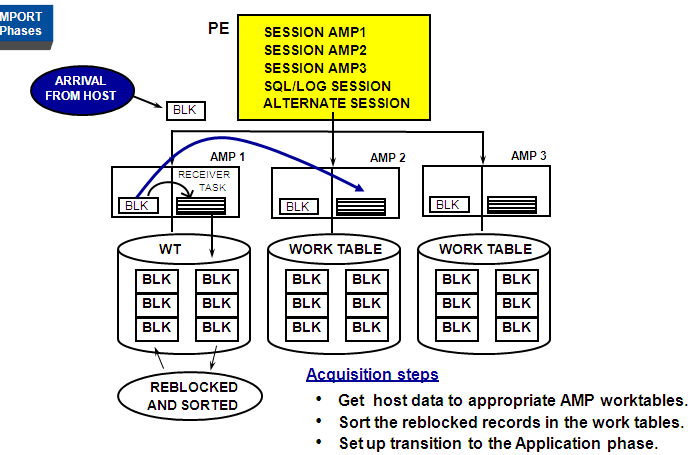
**Phase 3: Acquisition**

* 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**

****

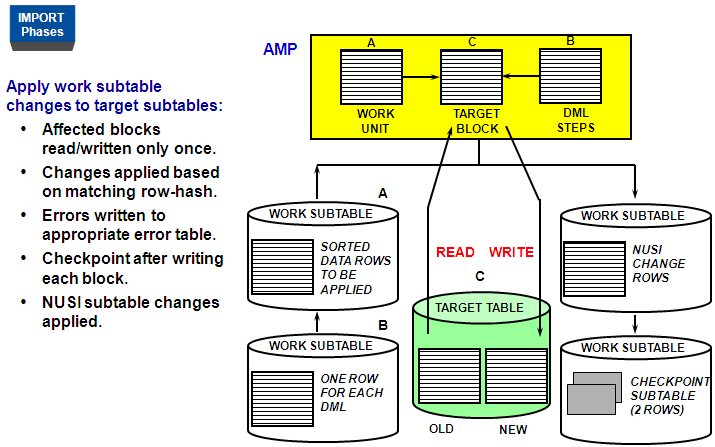
**Phase 4: Application**

* **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**

****

**Phase 5: Cleanup**

* **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.

**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**

**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.



**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**

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.

**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.

**52. 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 63 DML statements can be included for one IMPORT task.

Alternative to MultiLoad for low-volume batch maintenance of large databases; replacement for BulkLoad.

**Allows target tables to:**

Have secondary indexes and Referential Integrity constraints.

Be MULTISET or SET.

Be populated or empty.

Have triggers - invoked as necessary

Allows conditional processing.

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**

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.

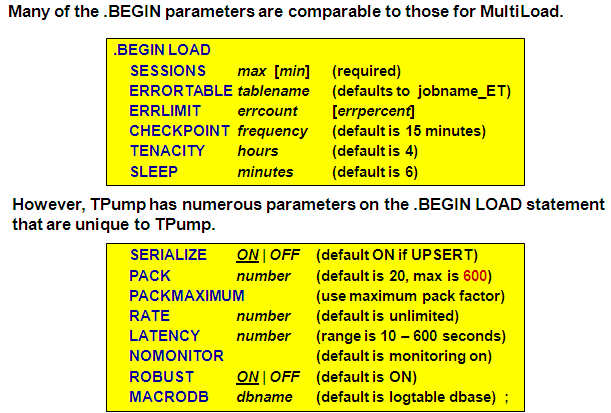
There is a limit of four IMPORT commands within a single TPump "load" task.

In using TPump with dates before 1900 or after 1999, the year portion of the date must be represented by four numerals (yyyy).

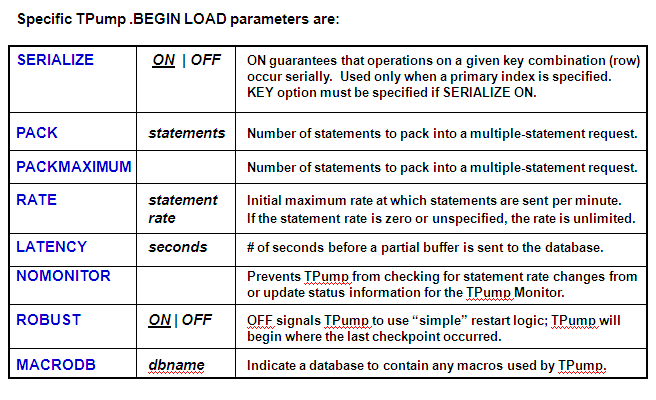
The default of two numerals (yy) to represent the year is interpreted to be the 20th century.

The correct date format must be specified at the time of table creation.

**.BEGIN LOAD Statement**

****

**TPump Specific Parameters**

****

**.**LOGTABLE restart\_log\_tpp;

.LOGON tdpid/username, password;

.BEGIN LOAD SESSIONS 4 SERIALIZE OFF

PACK 40 RATE 4800

ERRORTABLE Errors\_tpp ERRLIMIT 50 ;

.LAYOUT layout12;

.FIELD table\_code 1 CHAR(1);

.FIELD A\_Account\_Number 2 INTEGER;

.FIELD A\_Number \* INTEGER;

.FIELD A\_Street \* CHAR(25);

.FIELD A\_City \* CHAR(20);

.FIELD A\_State \* CHAR(2);

.FIELD A\_Zip\_Code \* INTEGER;

.FIELD A\_Balance\_Forward \* DECIMAL(10,2);

.FIELD A\_Balance\_Current \* DECIMAL (10,2);

.FIELD C\_Customer\_Number 2 INTEGER;

.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\_Account\_Number \* INTEGER;

.FIELD T\_Trans\_ID \* CHAR(4);

.FIELD T\_Trans\_Amount \* DECIMAL(10,2);

.DML LABEL lns\_Account;

INSERT INTO Accounts (account\_number, number, street, city, state, zip\_code, balance\_forward,

balance\_current )

VALUES ( :A\_Account\_Number, :A\_Number, :A\_Street, :A\_City, :A\_State, :A\_Zip\_Code,

:A\_Balance\_Forward, :A\_Balance\_Current );

.DML LABEL lns\_Trans;

INSERT INTO Trans (trans\_number, trans\_date, account\_number, trans\_id, trans\_amount)

VALUES ( :T\_Trans\_Number, :T\_Trans\_Date, :T\_Account\_Number, :T\_Trans\_Id, :T\_Trans\_Amount );

.DML LABEL lns\_Customer;

INSERT INTO Customer (customer\_number, last\_name, first\_name, social\_security)

VALUES ( :C\_Customer\_Number, :C\_Last\_Name, :C\_First\_Name, :C\_Social\_Security);

.IMPORT INFILE datafile1 LAYOUT layout12

APPLY lns\_Account WHERE table\_code = 'A'

APPLY lns\_Trans WHERE table\_code = 'T'

APPLY lns\_Customer WHERE table\_code = 'C';

.IMPORT INFILE datafile2 LAYOUT layout12

APPLY lns\_Account WHERE table\_code = 'A'

APPLY lns\_Trans WHERE table\_code = 'T'

APPLY lns\_Customer WHERE table\_code = 'C';

.END LOAD;

.LOGOFF;

**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.

Ex. of macro name - M2000216\_105642\_01\_0001

MultiLoad uses the DML statements.

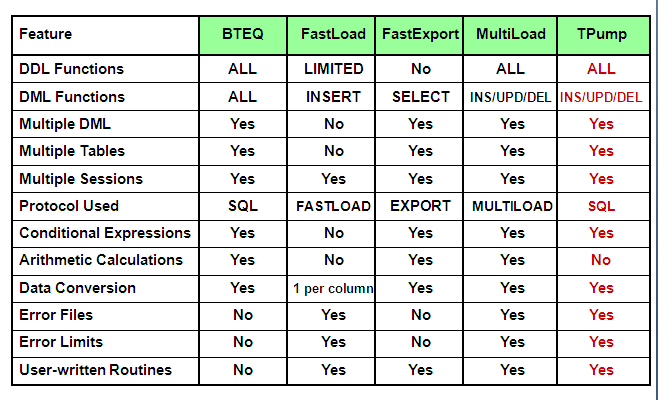
TPump uses row hash locking to allow for concurrent read and write access to target tables. It can be stopped with target tables fully accessible.

In Phase 4, MultiLoad locks tables for write access until it completes.

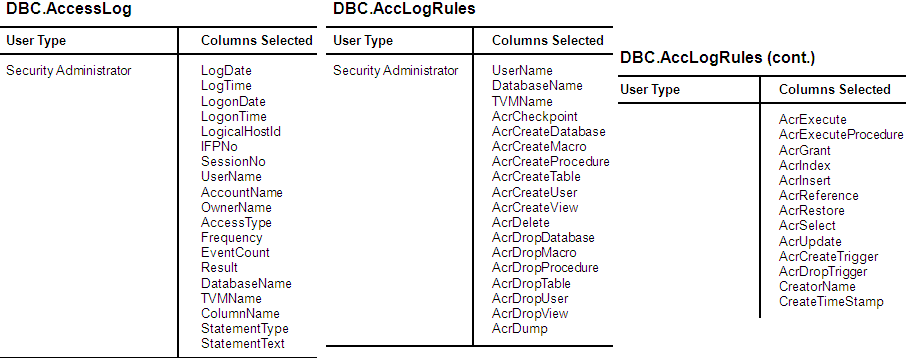
**Summary**

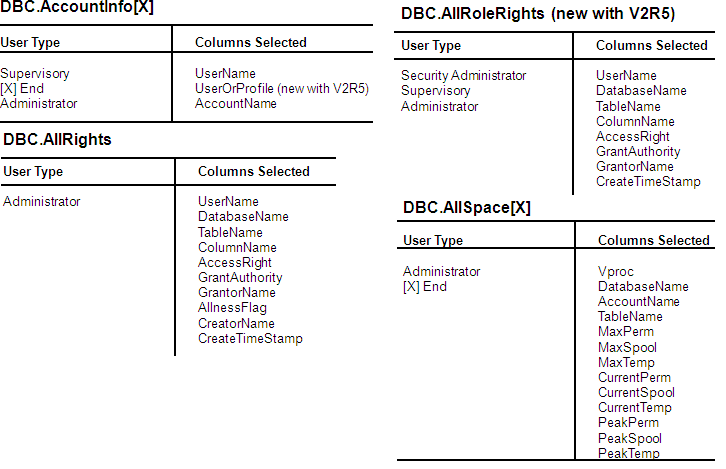
* Allows near real-time updates from transactional systems into the warehouse.
* Performs INSERTs, UPDATEs, and DELETEs to more than 60 tables at a time.
* Alternative to MultiLoad for low-batch maintenance of large databases; replacement for BulkLoad.
* Uses row-hash locks, allowing concurrent updates 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.

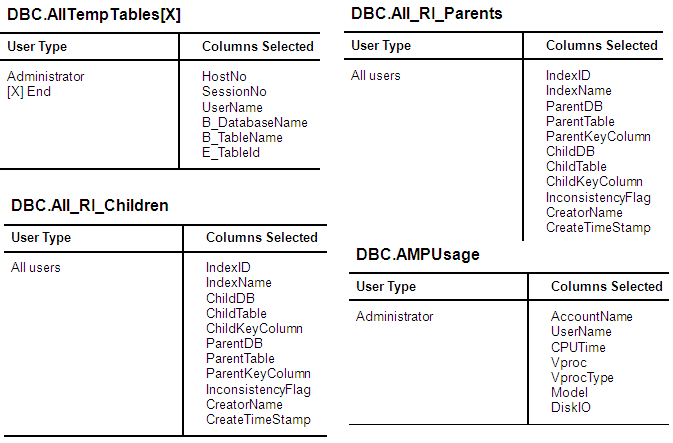
**Application Utility Checklist :-**

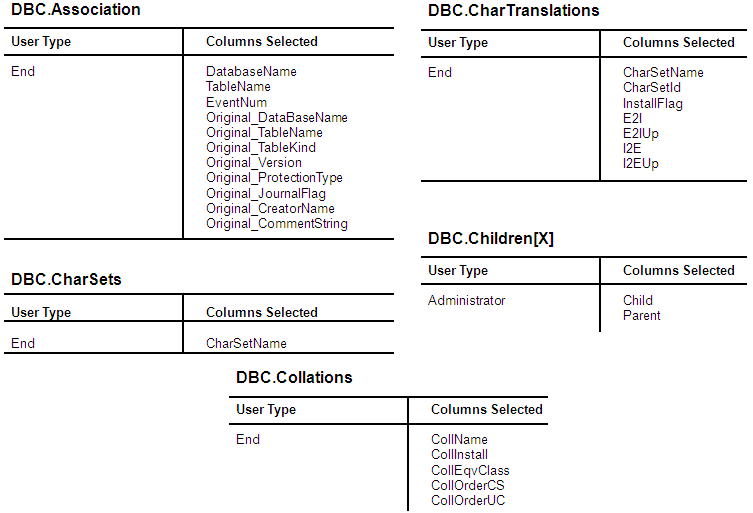
****

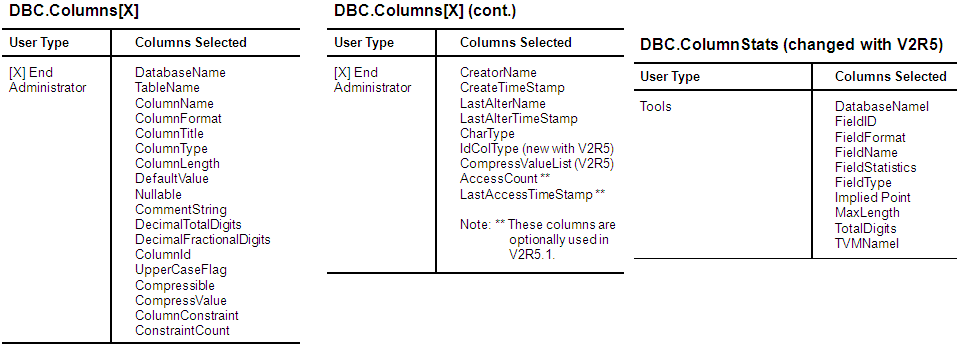
**53.** **Data Dictionary Views for V2R5.0 and V2R5.1 :-**

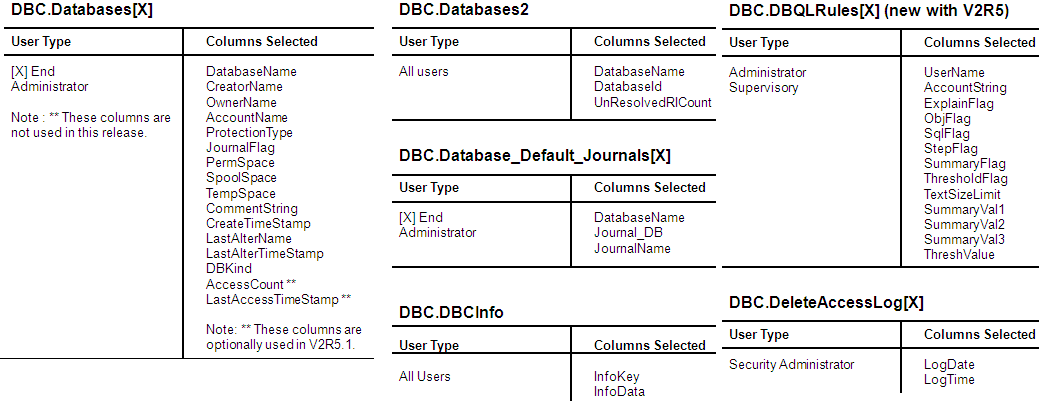


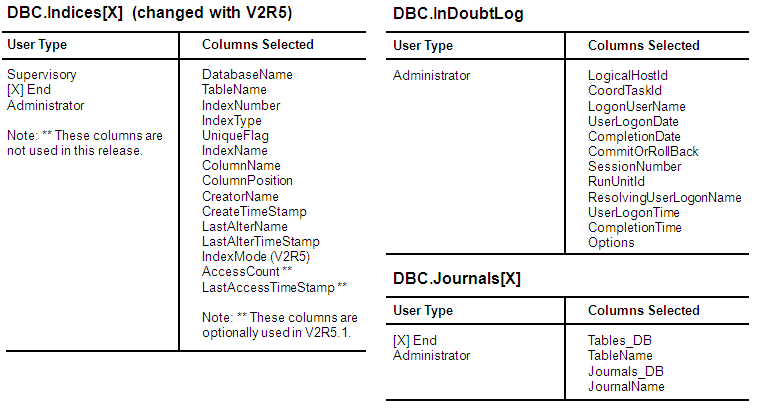


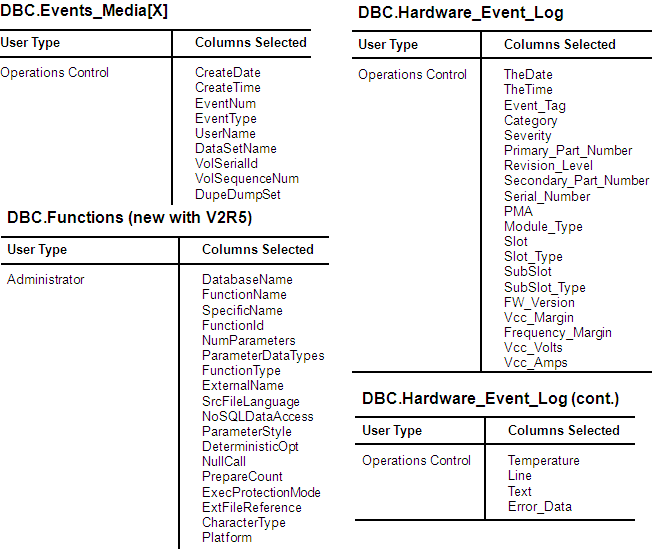


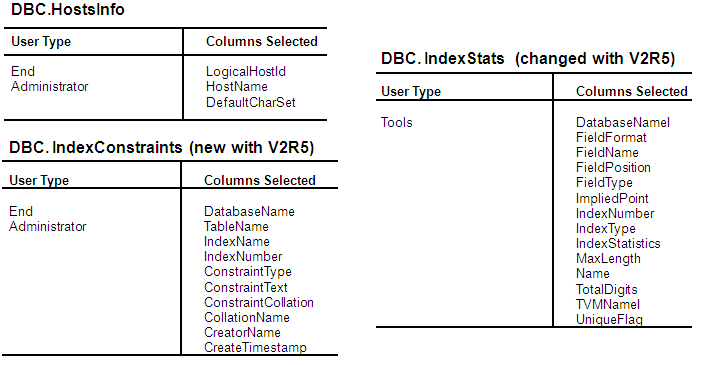


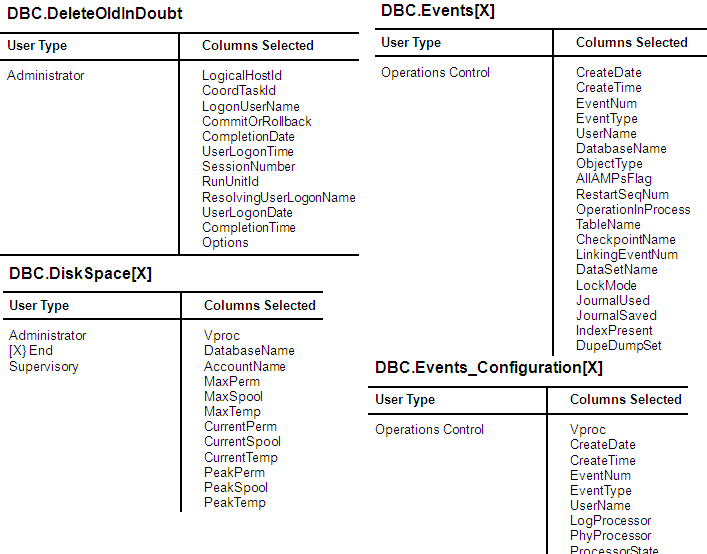
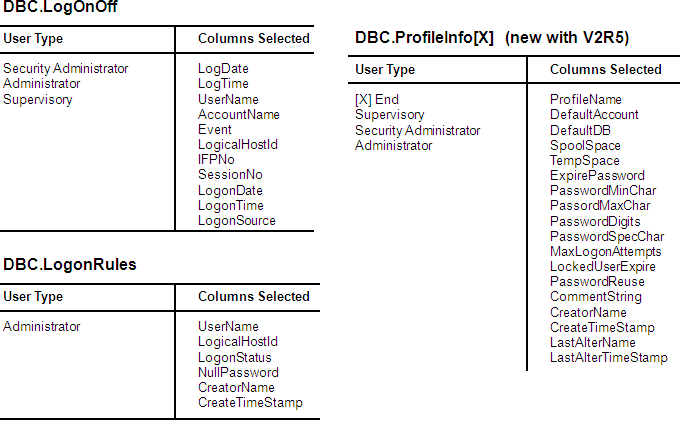


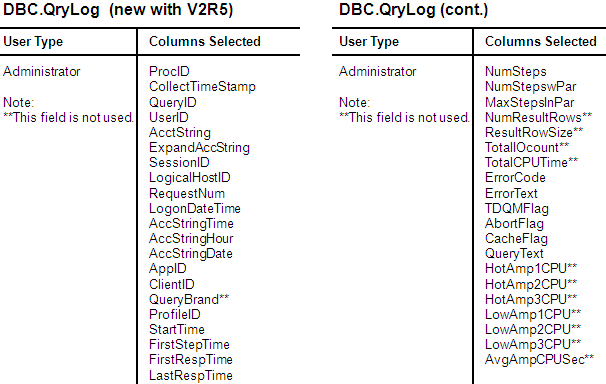


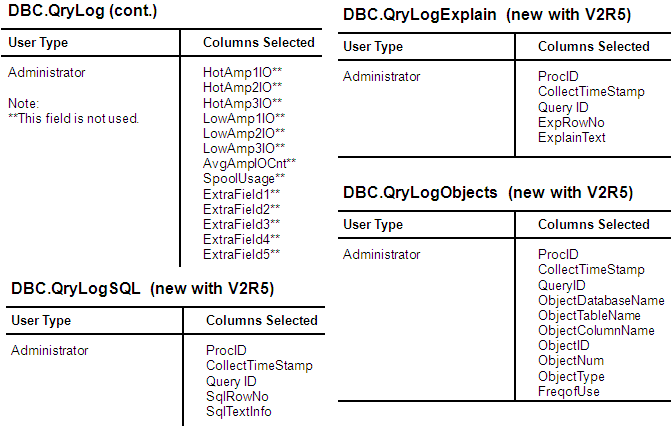


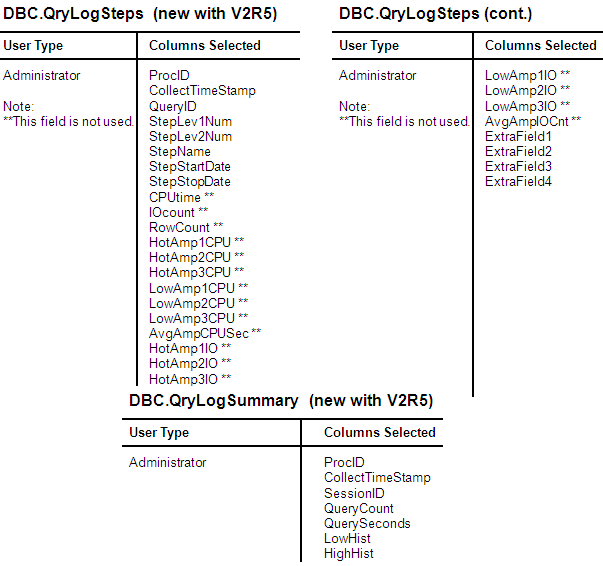


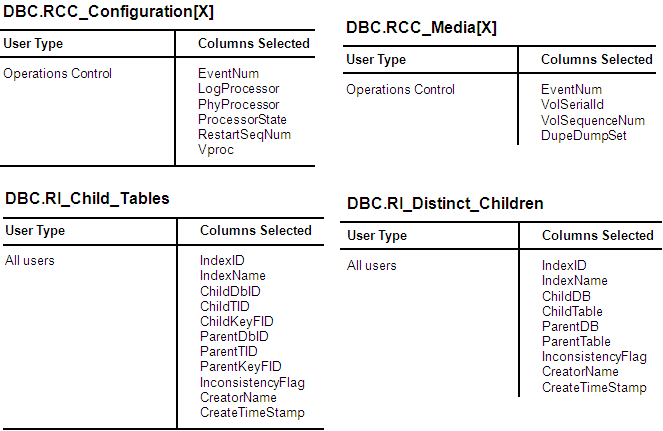


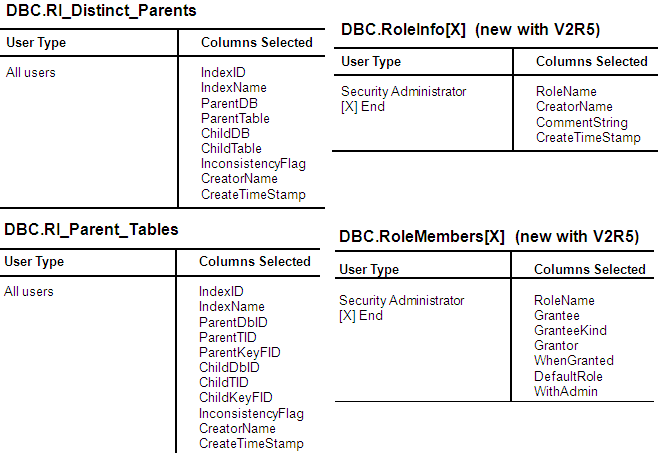


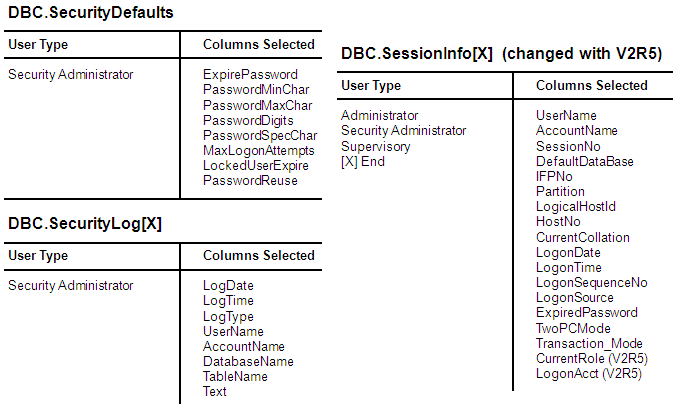


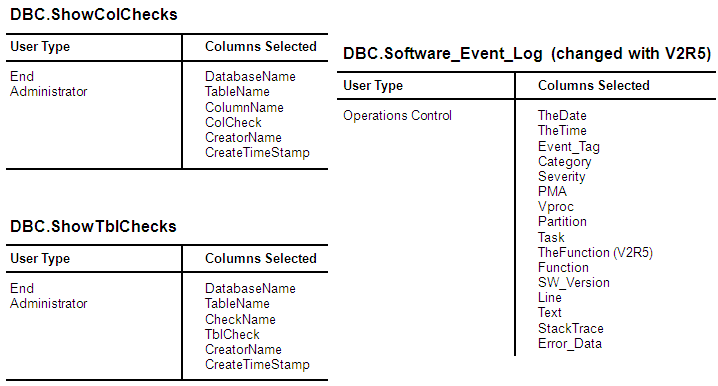


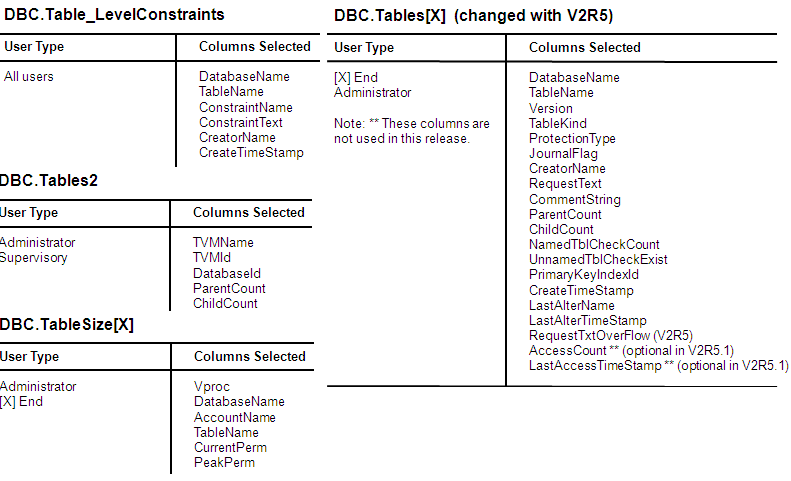


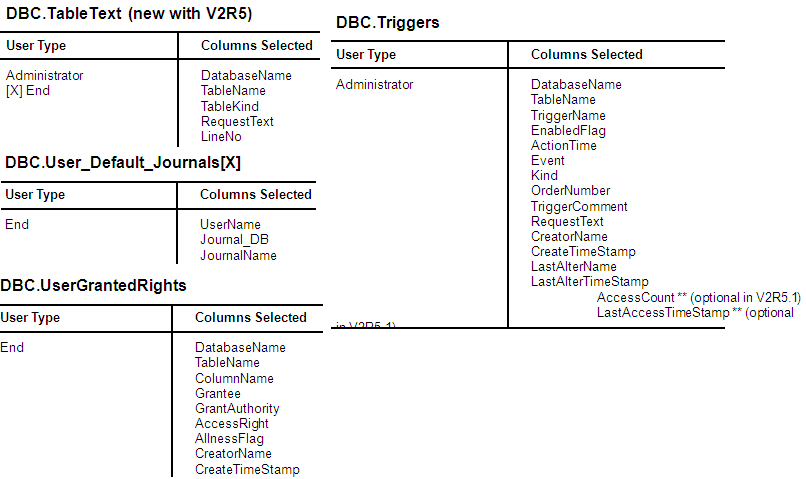


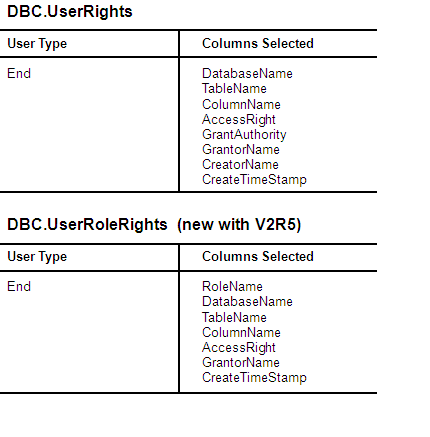


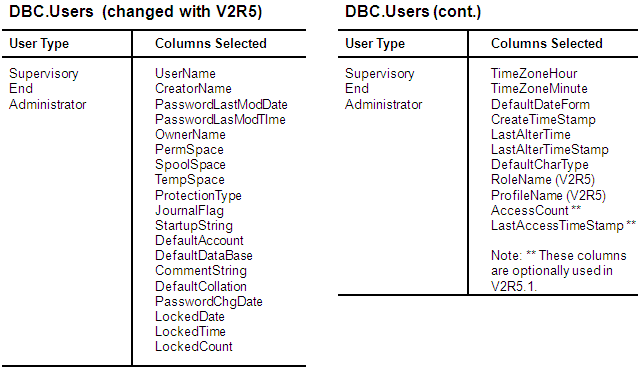












**54. Database Administration :-**

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, 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

Teradata Analyst Toolset – Index Wizard, Statistics Wizard, and Teradata SET

Teradata Dynamic Query Manager (DQM)

Database Console Window and System utilities – e.g., dbscontrol, ferret, rebuild, etc.

User scripts and 3rd party applications

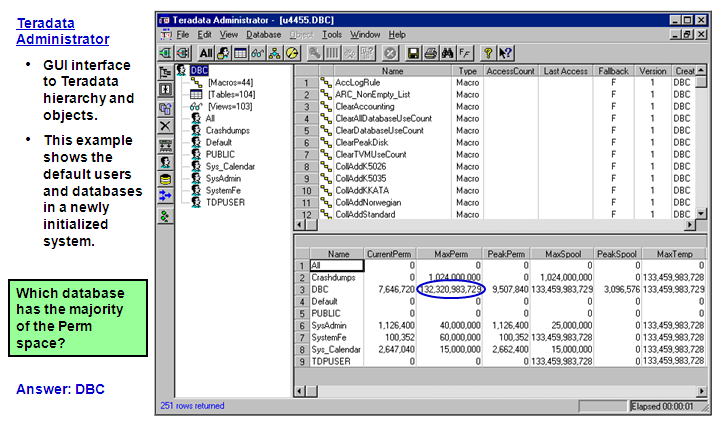
To do these functions, it is important to understand key concepts such as …

Teradata hierarchy and the concepts of ownership (parents and children)

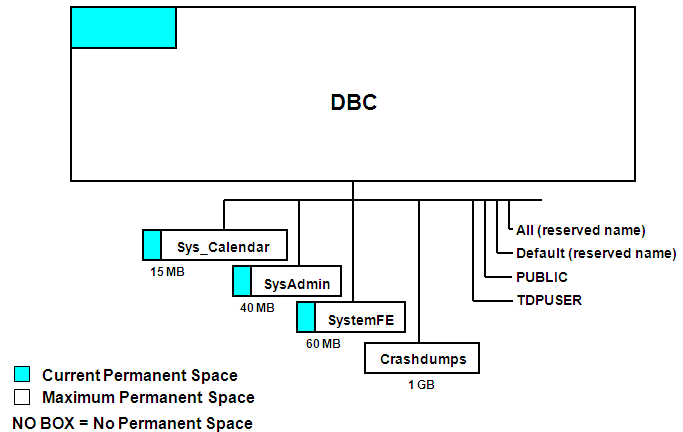
**55. Creating New Users and Databases :-**



**56. Teradata Administrator – New System :-**



**Initial Teradata Database :-**



**Summary :-**

* Initially, system user DBC owns all space in the Teradata Database except that owned by system users and databases SYS\_CALENDAR, SYSADMIN, SYSTEMFE, and CRASHDUMPS.
* 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 higher in the hierarchy is a *parent* or *owner*. Everyone 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 RDBMS Data Dictionary.

**57. Permanent Space Terminology :-**

**MaxPerm**

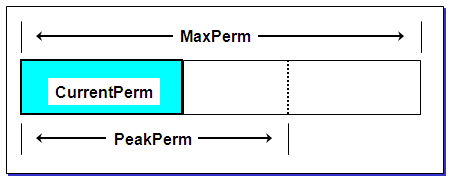
The maximum number of bytes available for table, index and permanent journal storage in a database or user.

**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.



**58. Spool Space Terminology :-**

**MaxSpool**

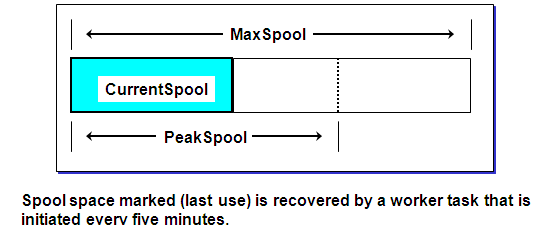
A value used to limit the number of bytes the system will consume to create spool files for a user.

**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.



**59. 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**

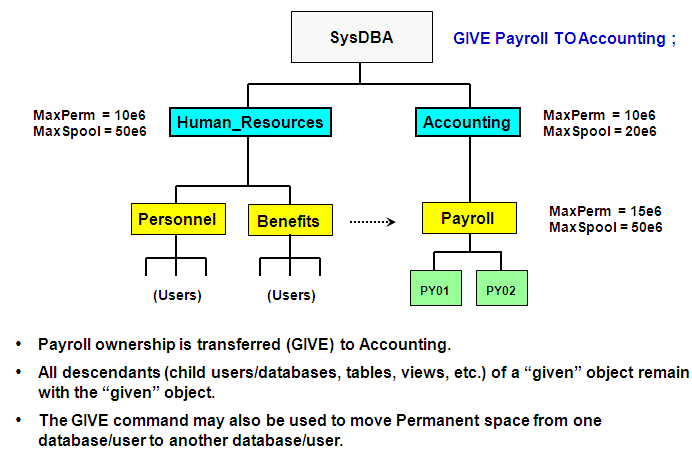
The number of bytes in use for global temporary tables.

**PeakTemp**

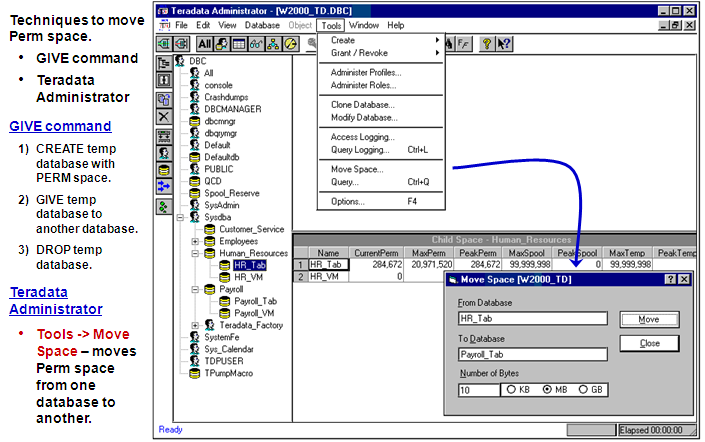
The maximum number of bytes used by global temporary tables for a user since the value was last reset.



**60.** **Giving One User to Another :-**



**61.** **Teradata Administrator – Move Space :-**



**62.** **Reserving Space for Spool :-**



**63. 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.

**64. 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 dis

tinct 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).

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.

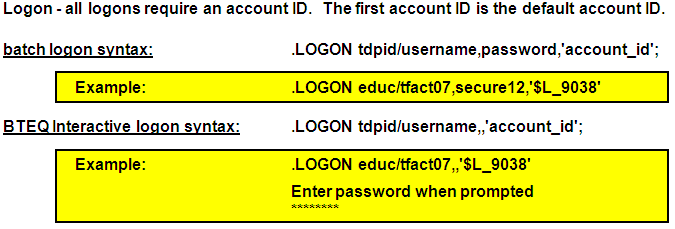
20) We spoil the parser's dictionary cache for the database.

21) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

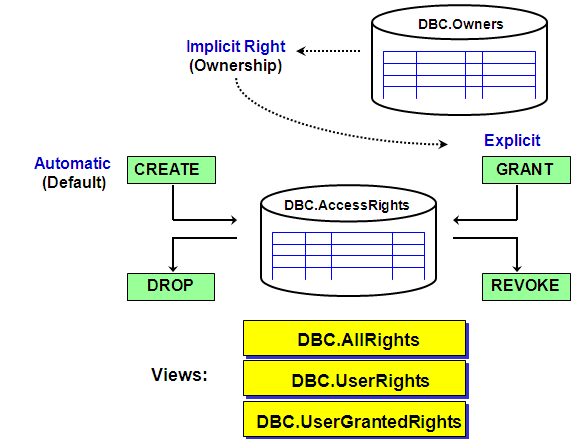
-> No rows are returned to the user as the result of statement 1.

**65.** **Creating and Using Account IDs :-**





**66. Access Rights Mechanisms :-**

****

**67. 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.AccessRights for write on a RowHash, we lock DBC.TVFields for write on a RowHash, we lock DBC.TVM for write on a RowHash, we lock DBC.DBase for read on a RowHash, and we lock DBC.Indexes for write on a RowHash.

4) We execute the following steps in parallel.

1) We do a single-AMP ABORT test from DBC.DBase by way of the unique primary index.

2) We do a single-AMP ABORT test from DBC.TVM by way of the unique primary index with no residual conditions.

3) We do an INSERT into DBC.TVFields (no lock required).

4) We do an INSERT into DBC.TVFields (no lock required).

5) We do an INSERT into DBC.TVFields (no lock required).

6) We do an INSERT into DBC.TVFields (no lock required).

7) We do an INSERT into DBC.Indexes (no lock required).

8) We do an INSERT into DBC.Indexes (no lock required).

9) We do an INSERT into DBC.TVM (no lock required).

10) 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.

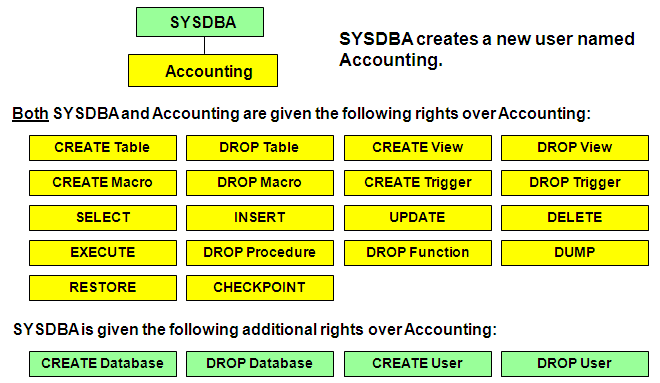
7) We modify the table header TFACT.Customer.

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.

**68. 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.



**69. 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.

**70. Roles and Profiles :-**

With Teradata V2R5, two new administration/security features are   
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.

**Access Rights Issues (prior to Roles) :-**

**The problems:**

Assume a customer has a large user base.

Assume that different users require different access rights on different objects - probably located in different databases.

Example: 300 different access rights for 10,000 users; this results in over 3 million access rights in the AccessRights table.

If users are not granted privileges to all of the objects within a database, then access rights have to be maintained for each object in the database.

If a user changes job functions, changing access rights can become tedious.

**Prior to Teradata V2R5, possible solutions were ...**

1. Place users into different parent databases based on their access right requirements.

Use the ALL option of the GRANT statement to grant rights on the shared object(s) to a parent database.

2. Grant the rights to users individually – an administrative nightmare.

**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.

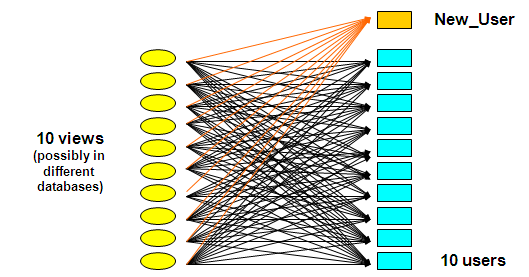
* The number of access rights in the DBC.AccessRights table is reduced.

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 :-**

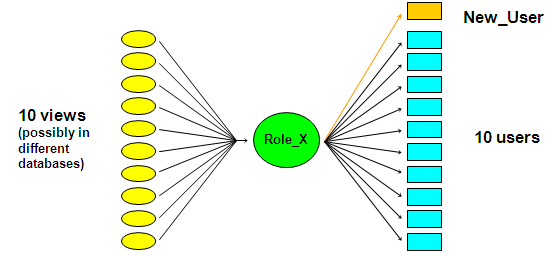
****

**GRANT SELECT ON View1, 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 :-**



First, create a role and grant privileges to the role.

**CREATE ROLE Role\_X;**

**GRANT SELECT ON View1, 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 the DBC.AccessRights table.

**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 Inquiry\_HR;

The 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 Inquiry\_HR;

**Current or Active Roles :-**

With V2R5.0, at any time, only one role will 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 = role\_name;**

A user may change roles by using …

**SET ROLE role\_name ;**

With V2R5.1, the SET ROLE ALL command allows a user to have all valid roles (for that user) to be active.

**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 :-**

**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 ;

**Example of Using Roles :-**

**Create roles.**

CREATE ROLE Inquiry\_HR;

CREATE ROLE Update\_HR;

**Assign access rights to the roles (partial listing).**

GRANT SELECT, EXECUTE ON HR\_VM TO Inquiry\_HR;

GRANT INSERT, UPDATE, DELETE ON HR\_VM TO Update\_HR;

**Grant users permission to use the roles.**

GRANT Inquiry\_HR TO Update\_HR; /\*nested role\*/

GRANT Inquiry\_HR TO Emp01, Emp02;

GRANT Update\_HR TO Emp03, Emp04;

GRANT Update\_HR TO Sup05 WITH ADMIN OPTION;

GRANT Batch\_HR\_Pay TO Sup05 WITH ADMIN OPTION;

**Modify the user to set the default role.**

MODIFY USER Emp01 AS DEFAULT ROLE = Inquiry\_HR;

MODIFY USER Emp02 AS DEFAULT ROLE = Inquiry\_HR;

MODIFY USER Emp03 AS DEFAULT ROLE = Update\_HR;

MODIFY USER Sup05 AS DEFAULT ROLE = Update\_HR;

**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 is 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[x]

**Example of Simplifying User Management :-**

**Example:**

* + **The problem:** 
    - A customer has 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 very 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:-**

**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 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;

MODIFY USER Emp02 AS PROFILE = Employee;

**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 PROFILE Example :-**

**Create a profile**

CREATE PROFILE Employee AS

ACCOUNT = '$M\_&D&H',

DEFAULT DATABASE = Personnel\_VMDB,

SPOOL = 100e6,

TEMPORARY = 50e6,

PASSWORD = (EXPIRE = 91, MINCHAR = 6, MAXLOGONATTEMPTS = 4,

LOCKEDUSEREXPIRE = 60, REUSE = 365);

**Assign the profile to a user.**

CREATE USER Emp01 AS PERM = 0,

PASSWORD = emp01pass,

PROFILE = Employee,

SPOOL = 200e6,

ACCOUNT = ('$M', '$H');

**What are the valid account code(s) of Emp01?**

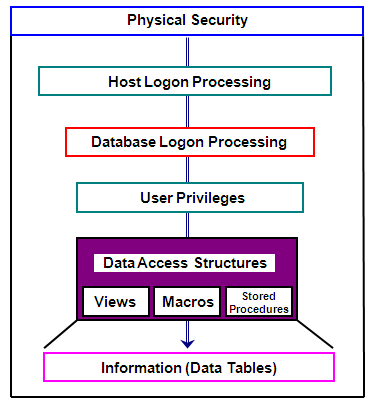
MODIFY USER Emp01 AS PROFILE = NULL;

What is the spool space limit for Emp01?

**Summary :-**

* A role is simply a collection of access rights.
  + Rights are first granted to a role and the right to use the role is   
    then granted to users.
  + **CREATE ROLE *role\_name*;**
* The GRANT and REVOKE commands have new extensions to support granting (or removing) access rights to roles.
* GRANT and REVOKE roles to users to simplify access right management.
  + **GRANT *role\_name* TO *user1*;**
* 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*;**

**71. System Access Control Levels :-**



**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) 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.

**Additional Utilities to View Sessions :-**

**QrySessn – system utility started via Database Console or Teradata Manager**

* + Provides display of active and idle sessions
  + Supervisor can be used to abort sessions

**xgtwglobal and gtwglobal – system utilities**

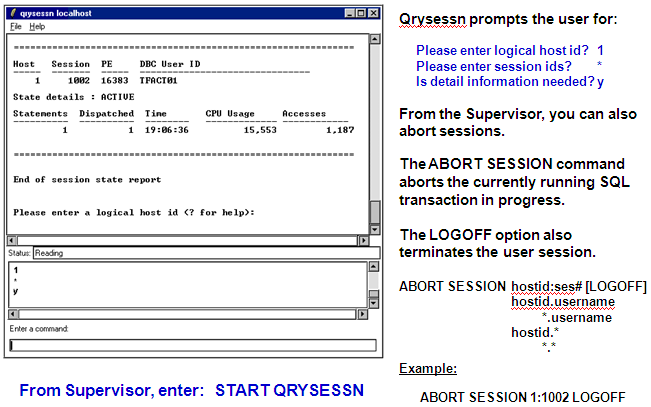
* + xgtwglobal – X windows (UNIX) utility that provides a GUI interface for gateway or LAN-based sessions
  + gtwglobal – command-line utility version which can be used with Windows or UNIX
  + Can be used to view and abort sessions

**Performance Monitor – Windows utility**

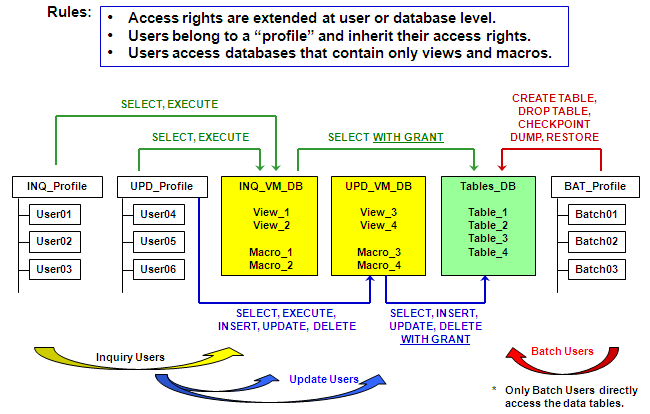
* + Can be used to view and abort sessions
  + Can be used to change a session’s priority
  + May be executed independently or via Teradata Manager

**–** **An example of Performance Monitor will be provided later in the course.**

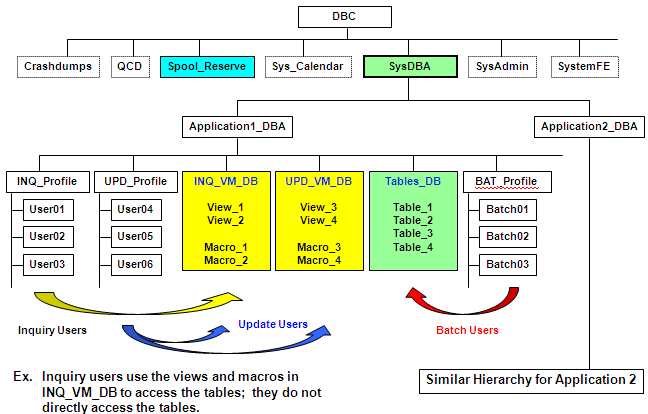
**Query Session Utility :-**

****

**A Recommended Access Rights Structure :-**

****

**A Recommended System Hierarchy :-**

****

**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.
* **System views you can use to monitor database access are DBC.SessionInfo, DBC.LogOnOff, and DBC.LogonRules.**
* **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 profile 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).

**72.** **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.
    - New facility available with V2R5.

**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.

**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

**Both facilities require establishing a set of rules.**

* + **Access Logging rules are stored in DBC.AccLogRules table.** 
    - Contains current access logging rules generated by BEGIN LOGGING and END LOGGING statements.
  + **DBQL (Query Logging) rules are stored in DBC.DBQLRuleTbl.** 
    - Contains current query logging rules generated by BEGIN QUERY LOGGING and END QUERY LOGGING statements.

**Both facilities provide a set of views to display logged information.**

**73. Implementing Referential Integrity :-**

**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.

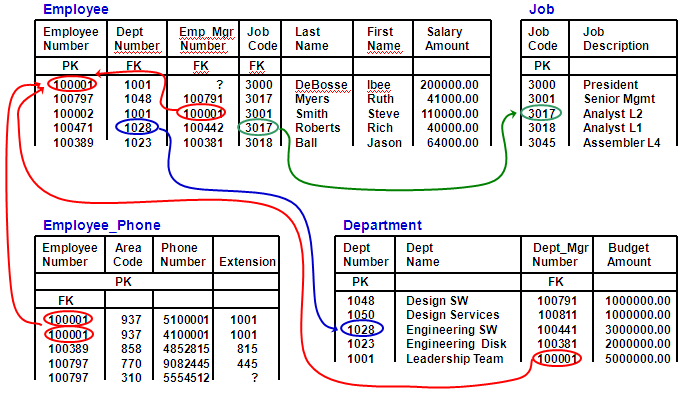
2. Create the Foreign Key references.

3. Populate the tables.

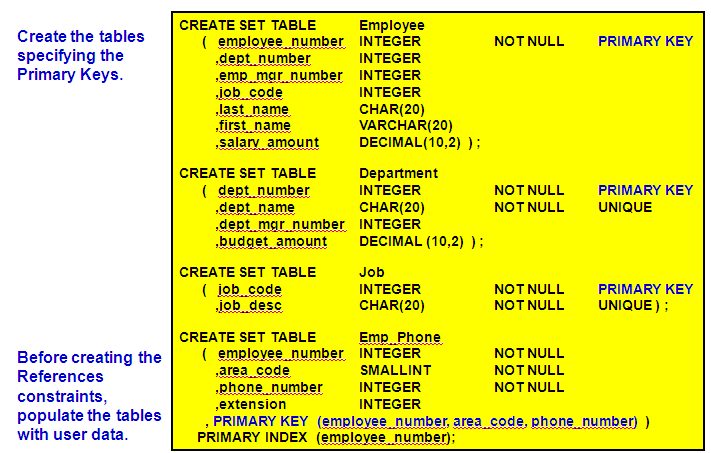
**Note:**

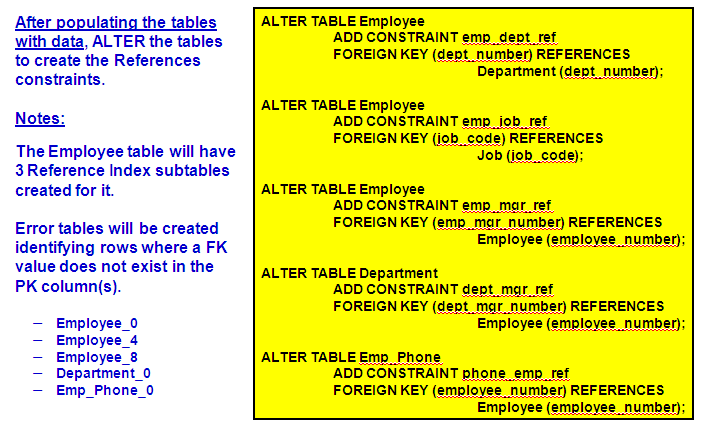
* + **Compress is not allowed on either the referencing or referenced columns. The data types must be the same.**

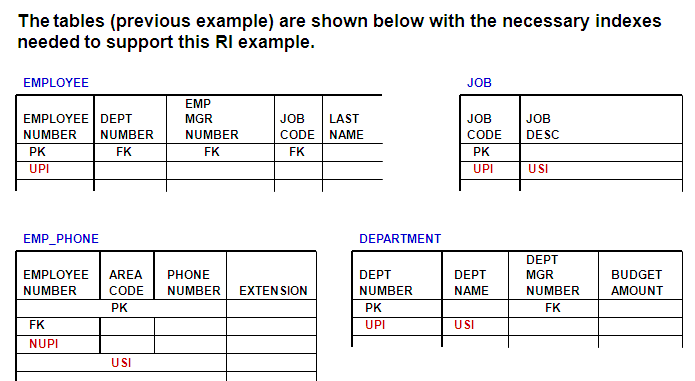
**Referential Model :-**

****

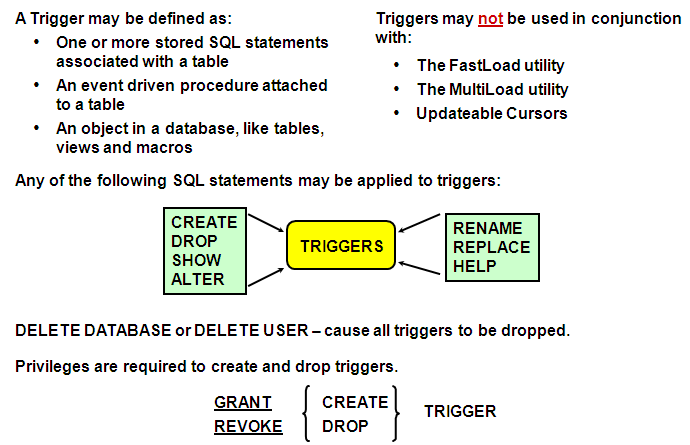
**Referential Integrity Example :-**

****

****

****

**What is a Trigger :-**

****

**Access Rights and Triggers :-**

**Example:** CREATE TRIGGER trigger1

AFTER UPDATE OF (col1) ON table1 FOR EACH ROW

WHEN NEW col1 > 100

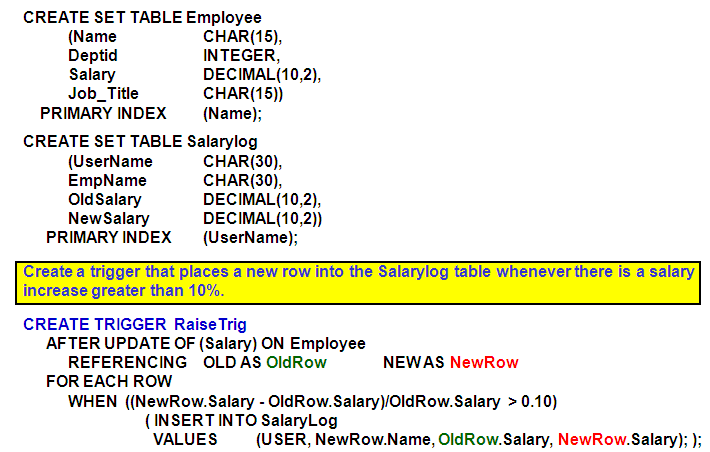
INSERT INTO log\_table VALUES ….

**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**

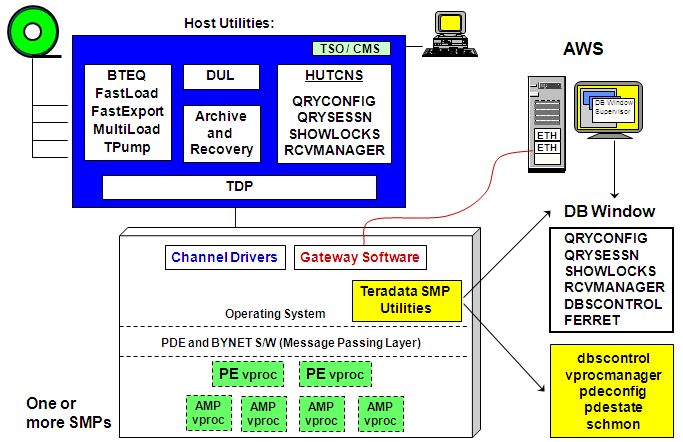
* + DROP TRIGGER privilege on the subject table or the database. The exception is when you use the REPLACE TRIGGER statement when no target trigger exists and you instead create a new trigger.
  + SELECT privilege on any column referenced in a WHEN clause or a triggered SQL statement subquery.
  + INSERT, UPDATE, or DELETE privileges on the triggered SQL statement target table, depending on the triggered SQL statement.

****

**Referential Integrity and Triggers Summary :-**

* Referential integrity is a concept of relationships between tables based on the definition of a primary key and a foreign key in the tables.
* Referential integrity prevents database corruption when application users execute INSERT, UPDATE and DELETE statements.
* When referential integrity is applied, columns within a referencing table are foreign keys for columns in another referenced table. You must define referenced columns as either:
  + Primary key columns, or UNIQUE columns
  + NOT NULL
* To create or replace triggers, you need specific privileges.

**74. Locations from which Utilities can be Executed :-**

****

**Summary**

* Teradata utilities are either host-based or AMP-based.
* The host-based utilities run under the host operating system and support user activities such as loading, backup and restore, and some maintenance functions for Teradata.
* **Examples include:**
  + Load utilities (FastLoad, MultiLoad, etc.)
  + Support utilities (pdestate, tpareset, etc.)
  + archive utilities (arcmain)
* **The AMP-based utilities run on the database and are accessed through the DBW.**
* **Examples include:**
  + Query Session (Qrysessn)
  + Configuration Display (Qryconfig)
  + Ferret Utility

**75. 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**

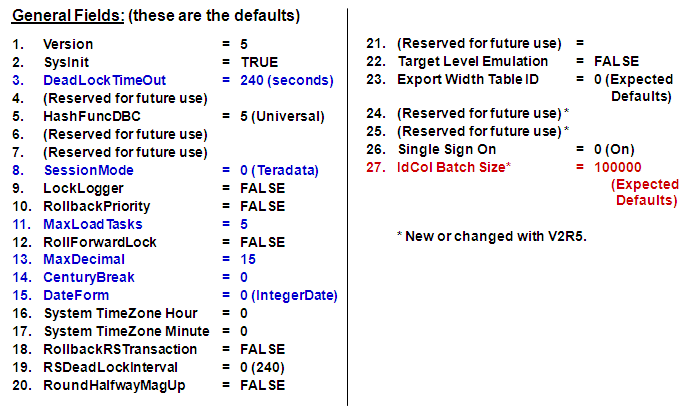
**The DBS Control Record parameters are divided into four categories:**

* + General
  + File System
  + Performance
  + Checksum (V2R5.1)

**DISPLAY or HELP commands**

* + DISPLAY GENERAL | FILESYS | PERFORMANCE | CHECKSUM
  + HELP GENERAL | FILESYS | PERFORMANCE | CHECKSUM

**DBS Control Record – General Fields (V2R5) :-**



**DBS Control Record – File System Fields :-**

**File System Fields:**

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. WriteDBsToDisk = FALSE

8. Cylinders Saved for PERM = 10 (cylinders)

Notes:

1.**FreeSpacePercen**t – 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.

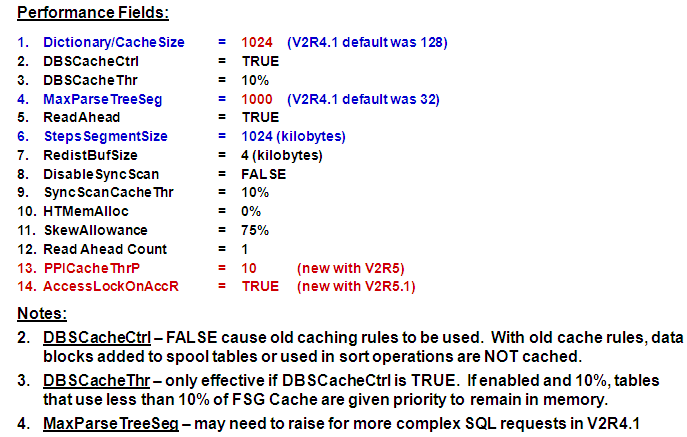
**3.PermDBSize** – sets default BlockSize

**6.PermDBAllocUnit** – number of sectors to allocate to new block - multi-row blocks are a multiple of this.

**7.WriteDBsToDisk** – forces writing of data blocks immediately to disk rather than committing to a backup node.

**8.Cylinders Saved for PERM** – cylinders not available to Spool.

**DBS Control Record – Performance Fields :-**

****

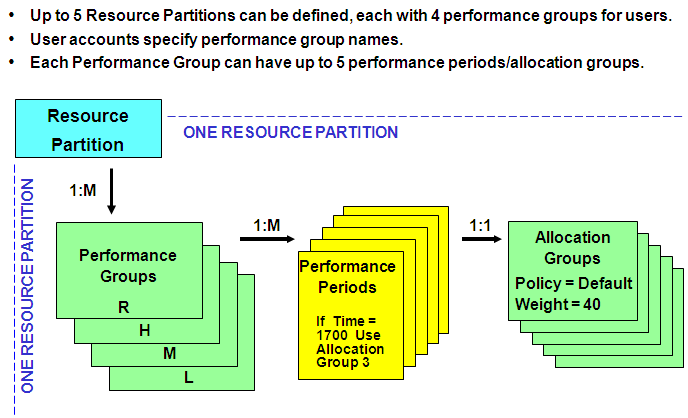
**Modifying DBS Control Parameters :-**

****

**Priority Scheduler Facility :-**

* **An integral part of the Teradata database, used internally by DB software.**
* **A tool for DBAs to mandate how database resources will be shared.**
* **Both CPU and Disk I/O are prioritized.**
* **Automatic change of priority if needed**
  + **Time of day**
  + **Resource usage**
* **All work in the database treated equal**
  + **Not biased toward the short and the quick SQL requests**
  + **No punishment for the lengthy**
* **Flexible: When activity is sparse, lower priority jobs get more resources.**
* **Teradata comes with Resource Partition 0 and priorities $L, $M, $H, and $R.**
* **Additional Resource Partitions (for a total of 5) can be defined.**
* **The schmon (or xschmon) utility is used to define additional Resource Partitions.**

**Priority Scheduler Concepts :-**

****

**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 :-**

**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 collection data and gather information from the operating system and the Teradata Database software.

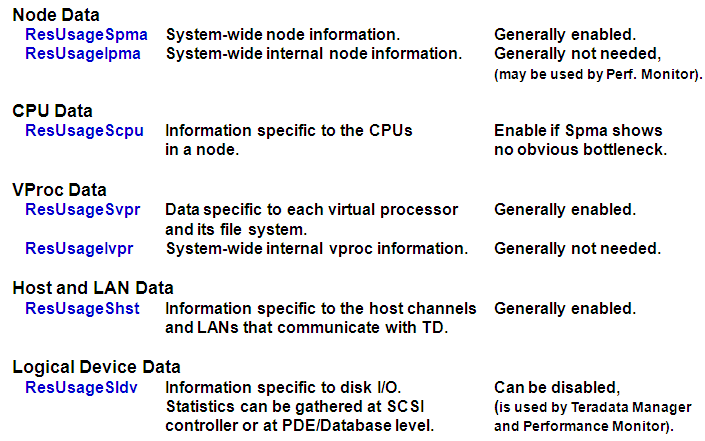
**Data logging**

* + RSS writes the collected data to ResUsage tables.

**Collections Costs**

* + Disk Space
  + Additional I/Os (minimal)
  + Additional CPU overhead (minimal)

**Resource Usage Tables :-**



**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**

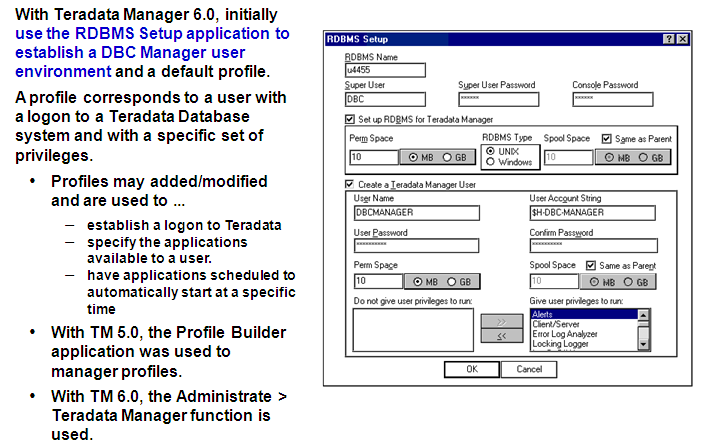
**Teradata Performance Monitor**

* + **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.
    - requires SVPR, SPMA, SCTL, IPMA, SLDV, SHST resource usage tables

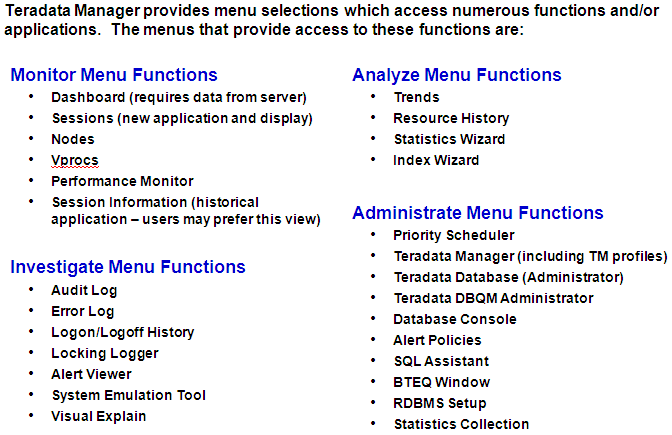
**76.** **Teradata Manager :-**

**Use Teradata Manager to:**

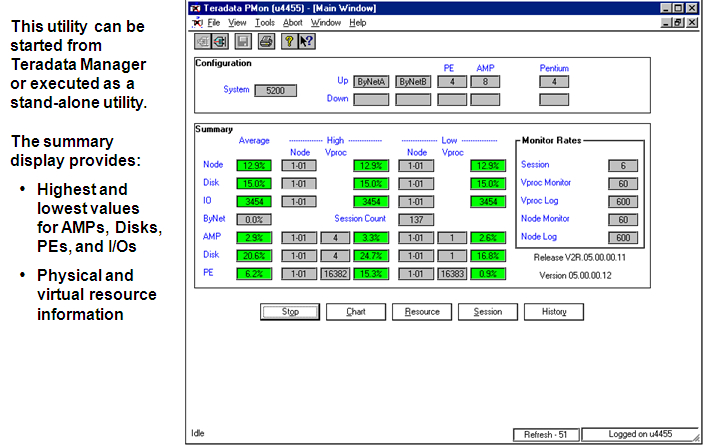
* + Provides special tools and applications to gather, manipulate, and analyze information about the Teradata Database
  + Provides an easy to use Graphical User Interface.
  + Can query status and utilization statistics.
  + Presents information in reports and graphs.
  + Can be used to administer Teradata Database users and sessions.
  + Can run several applications simultaneously.
  + Runs on a network-connected PC.



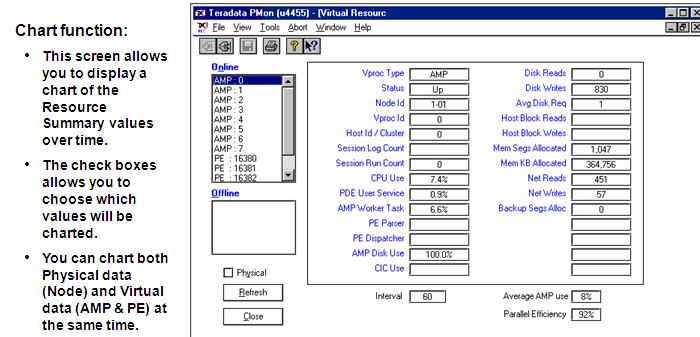
**Teradata Manager Applications :-**

****

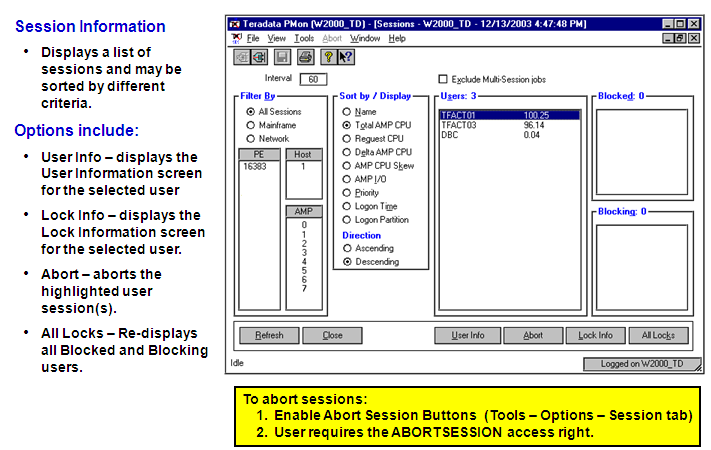
**Teradata Performance Monitor :-**

****

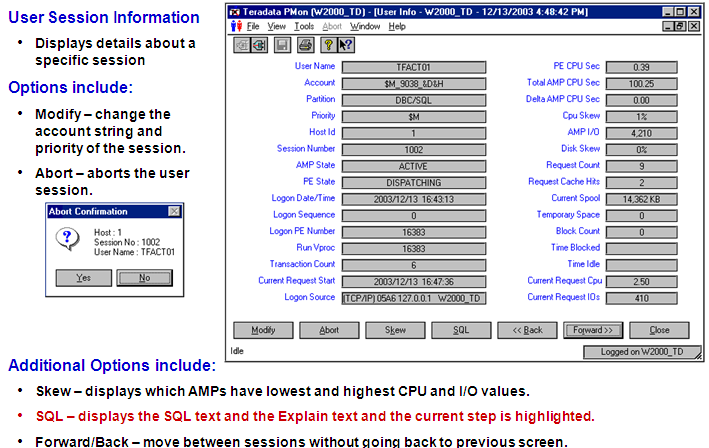
**Teradata Performance Monitor Resource Usage :-**

****

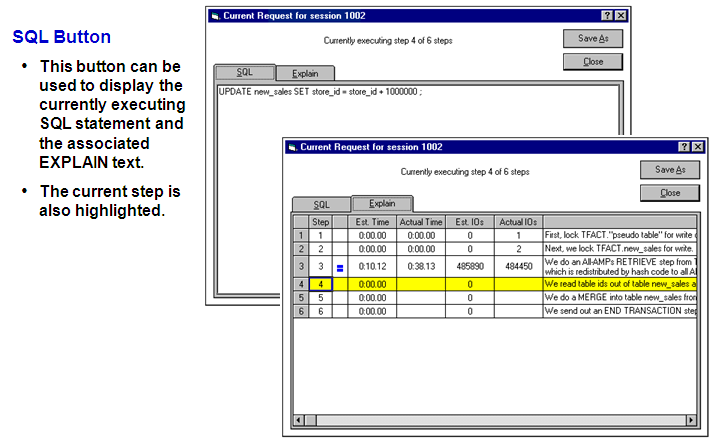
**Teradata Performance Monitor Session Information :-**

****

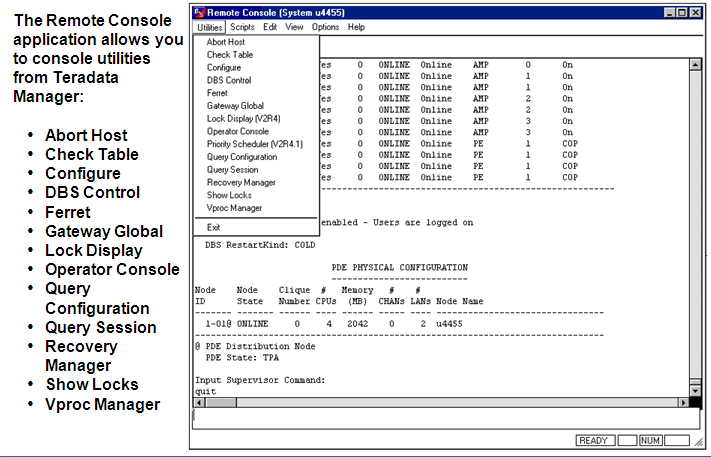
**Teradata Performance Monitor User Session Info :-**

****

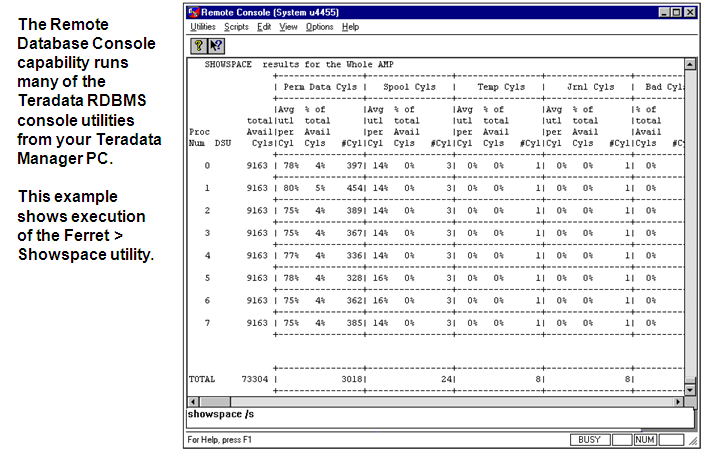
**Teradata Performance Monitor SQL & Explain Steps :-**

****

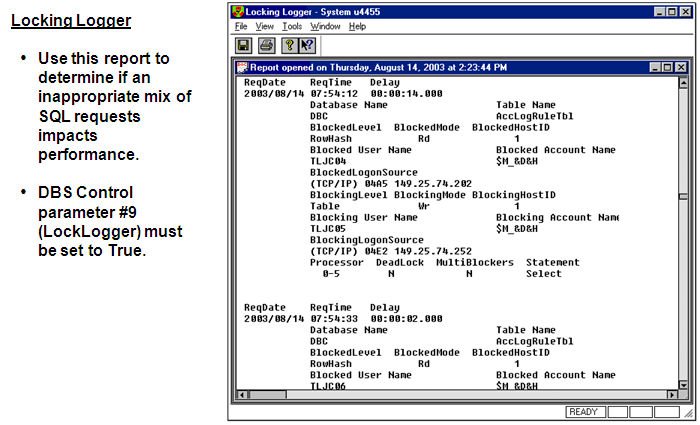
**Teradata Manager Remote Database Console :-**

****

**Teradata Manager Remote Database Console Example :-**

****

**Teradata Manager Locking Logger :-**

****

**Teradata Dynamic Query Manager (DQM) :-**

**The Teradata Dynamic Query Manager (DQM) product is the successor to the Database Query Manager (DBQM).**

This products provides the following capabilities:

1. Query Management: limits the execution of some queries on Teradata

2. Scheduled Requests: schedules SQL requests for batch execution

With Teradata DQM, the Query Management (QM) functionality …

* + Determines whether logon and query requests will be accepted by the Teradata database based on a set of user-defined “rules”.
  + Allows the processing of logon and query requests from all types of clients sources without any client software requirements.

The Teradata DQM Scheduled Requests (SR) capability …

* + Allows you to schedule requests through a TDQM scheduled request server.
  + This scheduled request server functions outside the Teradata RDBMS and is performed by TDQM Client and Server software packages.

Administration of Query Management and Scheduled Requests …

* + is performed by a TDQM Administrator software package.

**Teradata DQM – Query Management :-**

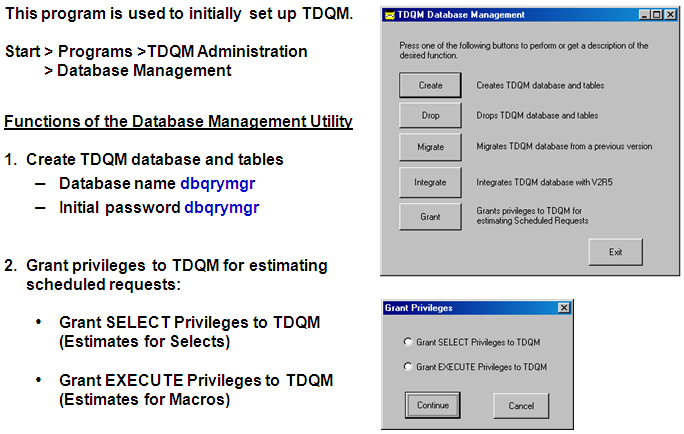
**What is Query Management?**

* + **Query Management is a set of user-defined rules that 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).
    - 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.
  + **Query Management has three Rule types:**
    - Object Access Rules (Date & Time each rule applies)
    - Query Resource Rules (Date & Time and Resource Limits)
    - WorkLoad Limits Rules (Date & time and Workload limits) ... New in V2R5

**Why use the Query Management facility?**

* + Enables the DBA to effectively manage access to and the use of Teradata resources.
  + TDQM addresses the key problems of database system overload and network saturation that result from a large number of clients accessing the Teradata RDBMS.

**Functions of the Database Management Utility :-**

****

**77. 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)
  + Hardware failure
  + Software failure
  + Accidents

**Restart Processes**

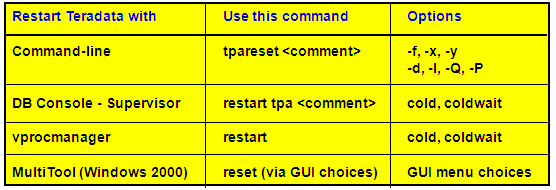
1. Spool cylinders are returned to free cylinder list (unused cylinder pool).

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 :-**



**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;

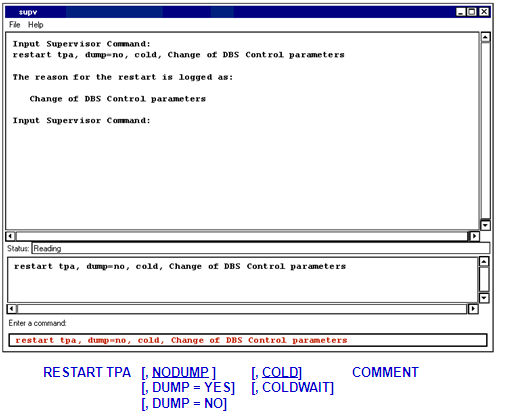
EXEC ALLRESTARTS (DATE - 7,);

LOGOFF;

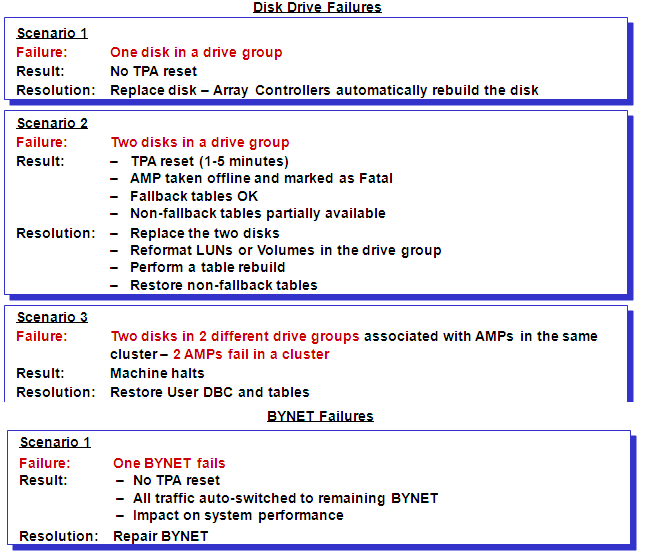
**The “tpatrace” command may also be used to see information about restarts.**

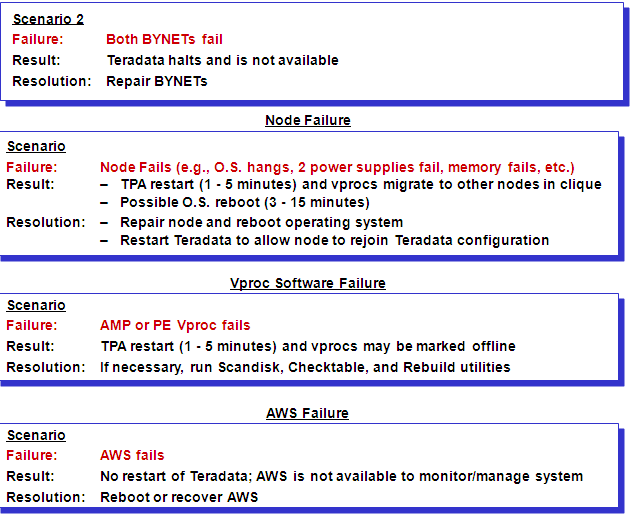
# tpatrace 3 (shows last 3 restarts)

**Restarting Teradata from DB Window :-**



**Unscheduled Restarts :-**





**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)**

**78. Ferret – Defragment and Packdisk :-**

**DEFRAGMENT :**

Combines free sectors and moves them to the end of a cylinder.

**PACKDISK :**

fill (packs) cylinders up to the Free Space Percentage (FSP).

**Checking Data Integrity :-**

**SCANDISK**

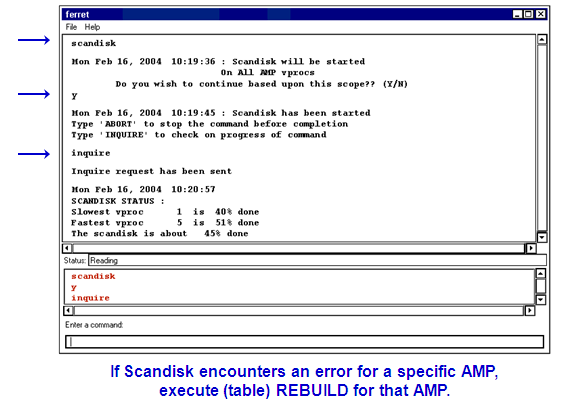
Checks the AMP’s file system structures (CIs and DBs for consistency)

**CHECKTABLE**

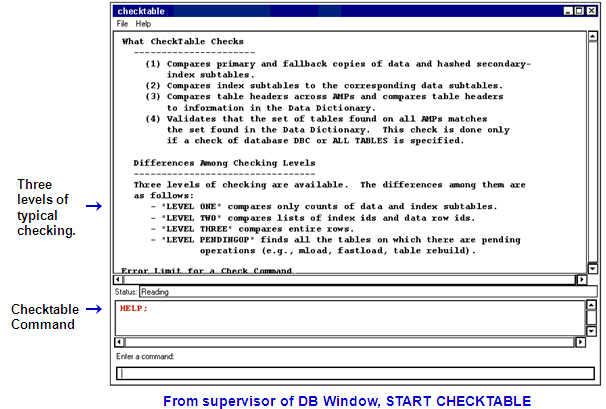
Checks for consistency in internal data structures such as table headers, SI subtables, row identifiers, etc.

**Typically, first execute SCANDISK, then CHECKTABLE.**

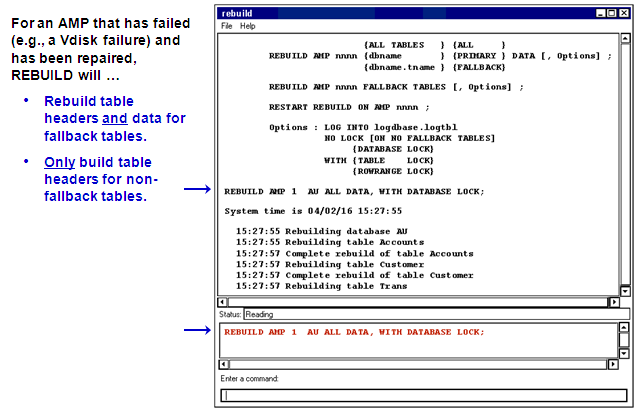
**Ferret – Scandisk Utility :-**



**Checktable Utility :-**



**Table Rebuild Utility :-**



**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.

**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 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 :-**

**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 in V2R5.1 include:**

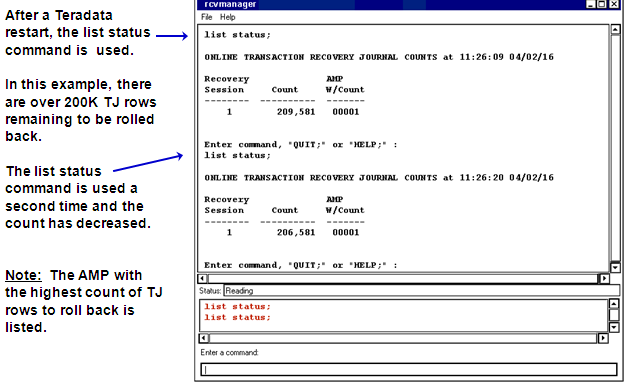
LIST ROLLBACK TABLES; – view the tables being rolled back for which rollback processing is pending cancellation

LIST CANCEL ROLLBACK TABLES; view the tables that are pending cancellation

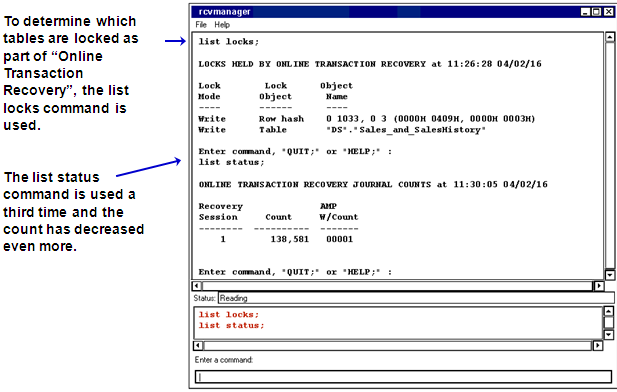
CANCEL ROLLBACK ON TABLE <table-id> [{, <table-id>} ...] ;

ROLLBACK SESSION <host>, <session> PERFORMANCE GROUP [<Perf Group Name>];

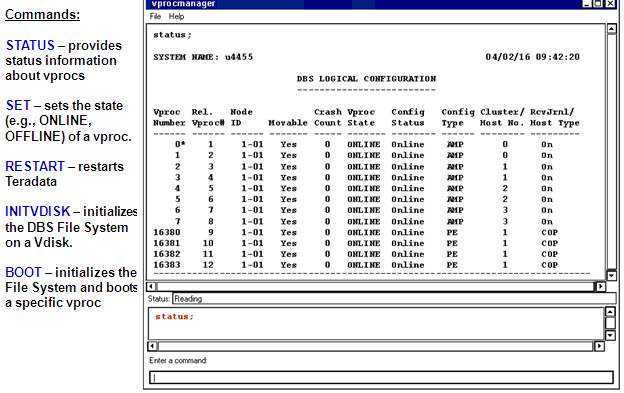
**Rcvmanager – List Status :-**



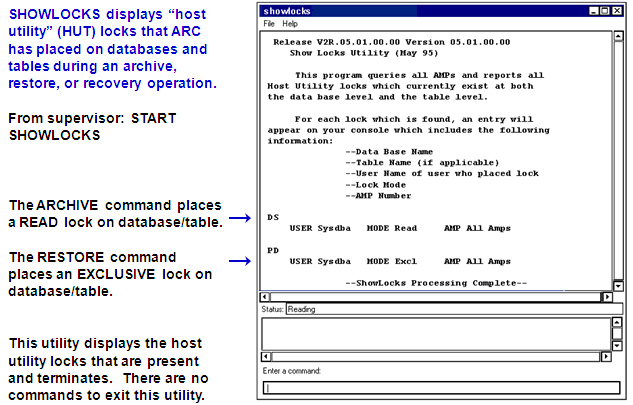
**Rcvmanager – List Locks :-**

****

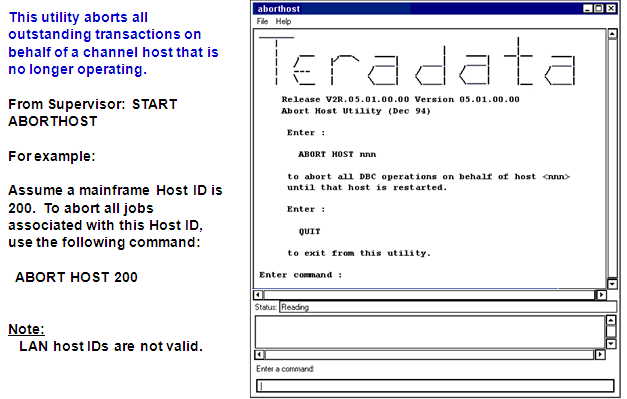
**Vprocmanager :-**



**Showlocks Utility :-**



**Abort Host Utility :-**



**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.
* Abort Host – aborts all outstanding transactions on behalf of a channel host that is no longer operating.

**79. 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**

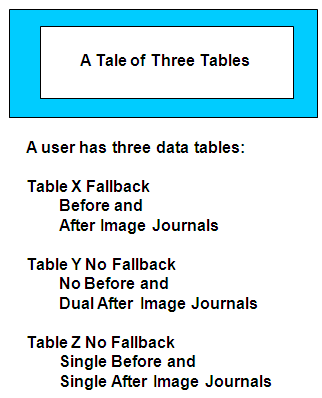
**Permanent journals:**

* + Optional features that can provide protection for software and hardware failures.
  + Store committed, uncommitted and aborted changes.
  + Users manage journal tables.

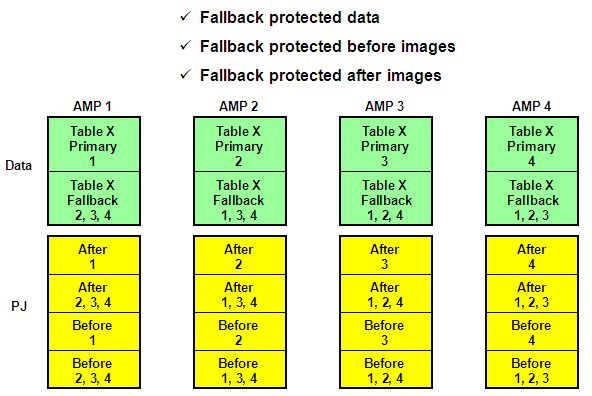
**Permanent journal options:**

* + 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.

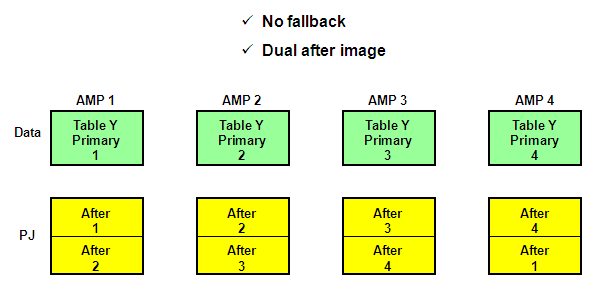
**Permanent Journal Scenario :-**

****

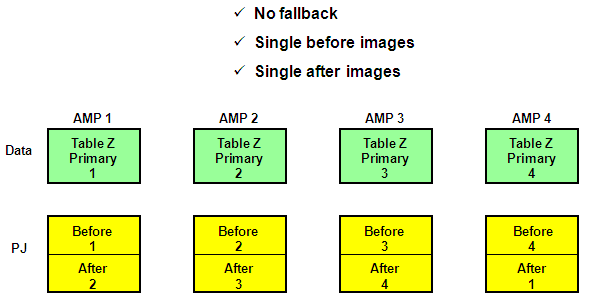
**Table X :-**

****

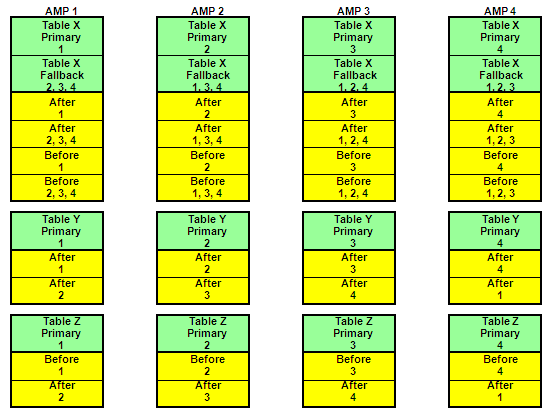
**Table Y :-**

****

**Table Z :-**

****

**Permanent Journals (Putting all tables three together) :-**

****

**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 :-**

**Monday:**

**Archive journals X, Y and Z**

**Archive table X**

**Tuesday:**

**Archive journals X, Y and Z**

**Archive table Y**

**Wednesday:**

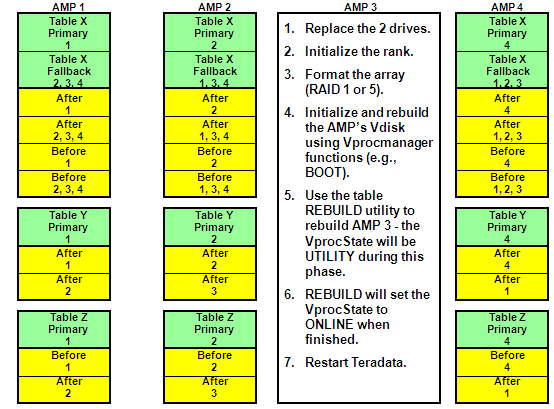
**Archive journals X, Y and Z**

**Archive table Z**

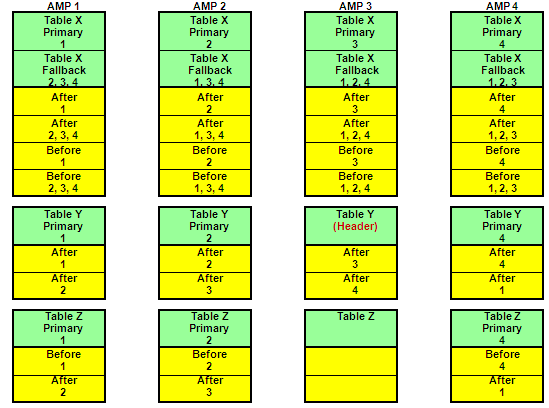
**Thursday:**

**AMP 3: Two drives fail in a drive group**

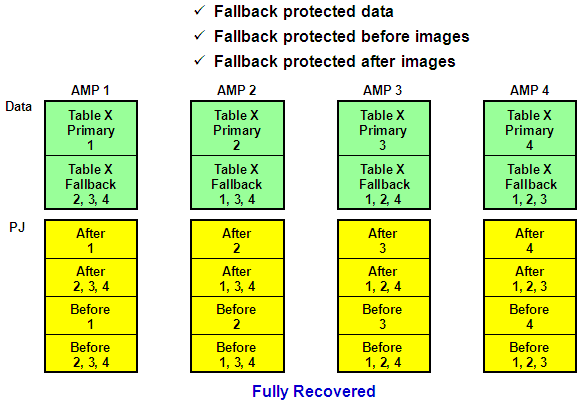
**After Restart Processing Completes :-**

****

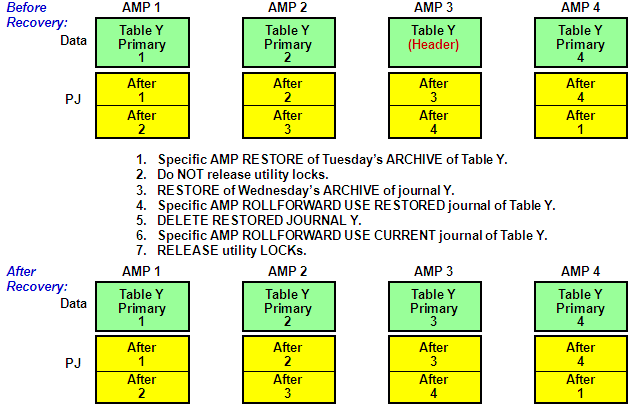
**After REBUILD and Restart of Teradata :-**

****

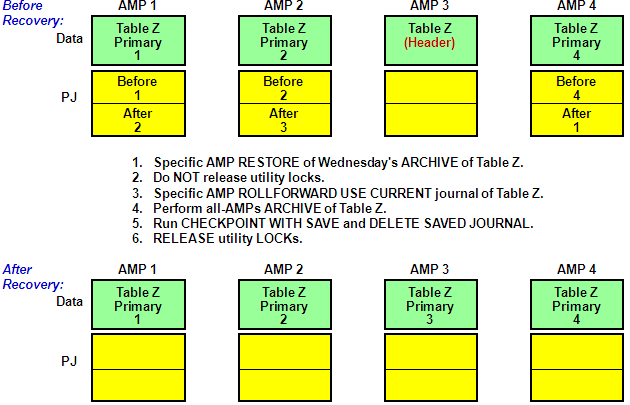
**Table X Recovery :-**

****

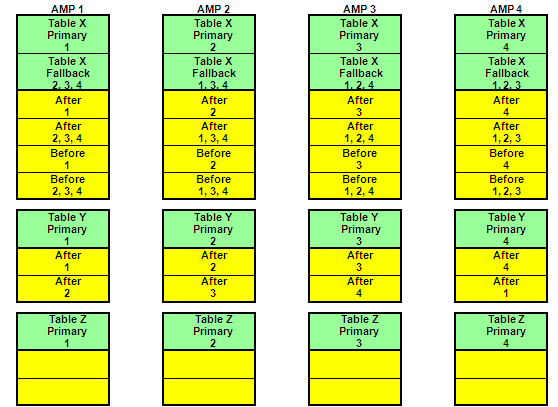
**Table Y Recovery :-**

****

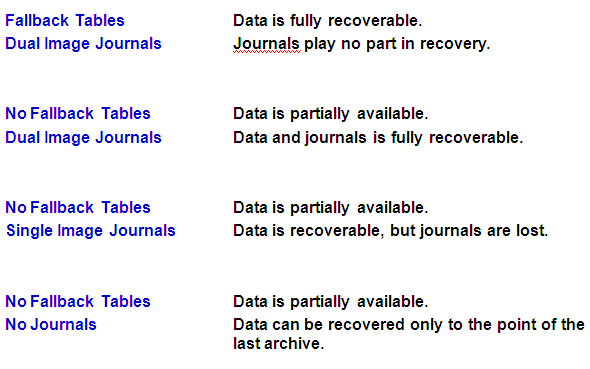
**Table Z Recovery :-**

****

**After Recovery :-**

****

**Summary :-**

****

**80. Archive and Recovery Utility (ARC)**

**Archive**

* + Captures user data on portable storage media

**Restore**

* + Restores data from portable storage media

**Recovery**

* + Recovers changes to data from permanent journal tables.

**Common uses of ARC**

* + 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
  + Moving tables from 1 system to another (e.g., moving tables from test system to a production system)

**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** :-

**FastLoad (3 Tables - 3 jobs)**

**or**

**RESTORE (3 Tables - 1 job)**

**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 Version 7.0 :-**

**ARC Version 7.0 is required to archive/restore Teradata V2R5.**

* + Provides archive, restore, and copy functionality for V2R5.0
  + Provides for data migration between V2R3/V2R4.0/V2R4.1 to V2R5.0.
  + Required dependency of the Open Teradata backup products.

**There are several ways to invoke the Archive facility.**

* + NetVault (from BakBone software) – Version 6.5.3
  + NetBackup (from VERITAS software) – limited support
  + ASF2 (previous X Windows tape management utility - not supported with V2R5)
    - ASF2 Tape Reader 2.0 can be used to read previous ASF2 tapes
  + Command Line (execute arcmain)
  + Host or Mainframe

**Utilities such as NetVault allow you to create scripts, schedule jobs, and provide various tape management capabilities.**

**ARCHIVE :-**

|  |
| --- |
| **LOGON dbc/sysdba,dbapass;**  **ARCHIVE DATA TABLES (PD)**  **(EXCLUDE TABLES (dept\_summary, phone\_summary))**  **, ABORT**  **, RELEASE LOCK**  **, FILE = arch2\_PD;**  **LOGOFF;** |

|  |
| --- |
| **LOGON dbc/sysdba,dba**  **pass;**  **ARCHIVE DATA TABLES (PD)**  **, ABORT**  **, RELEASE LOCK**  **, FILE = arch1\_PD;**  **LOGOFF;** |

|  |
| --- |
| **LOGON dbc/sysdba,dbapass;**  **ARCHIVE DATA TABLES (Sysdba) ALL**  **, ABORT , RELEASE LOCK**  **, FILE = arch3\_Sys;**  **LOGOFF;** |

**This script archives Sysdba and all of its child databases/users.**

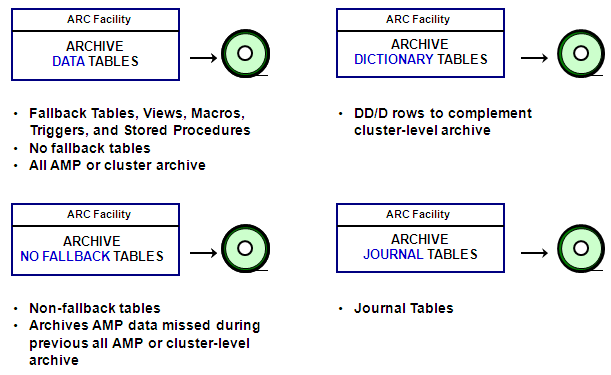
|  |
| --- |
| **LOGON dbc/sysdba,dbapass;**  **ARCHIVE DATA TABLES**  **(Sysdba) ALL**  **(EXCLUDE TABLES (PD.dept\_summary, PD.phone\_summary))**  **, EXCLUDE (Demo), (Guest\_Users) ALL, (Sandbox)**  **, ABORT , RELEASE LOCK**  **, FILE = arch4\_Sys;**  **LOGOFF;** |

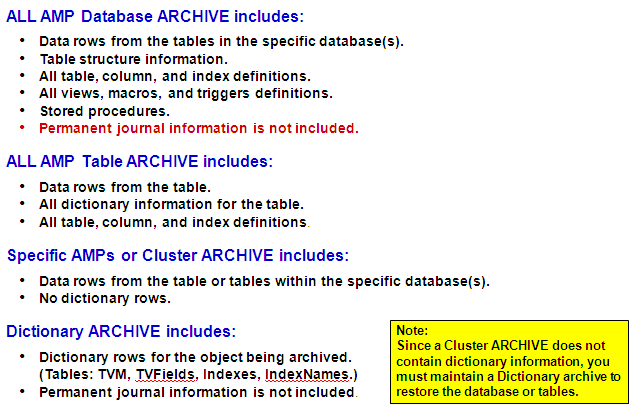
**This script archives Sysdba and all of its child databases/users and excludes some tables and databases**

|  |
| --- |
| **LOGON dbc/dbc,dbcpass;**  **ARCHIVE DATA TABLES (DBC) ALL**  **, ABORT, RELEASE LOCK**  **, FILE = arch5\_DBC;**  **LOGOFF;** |

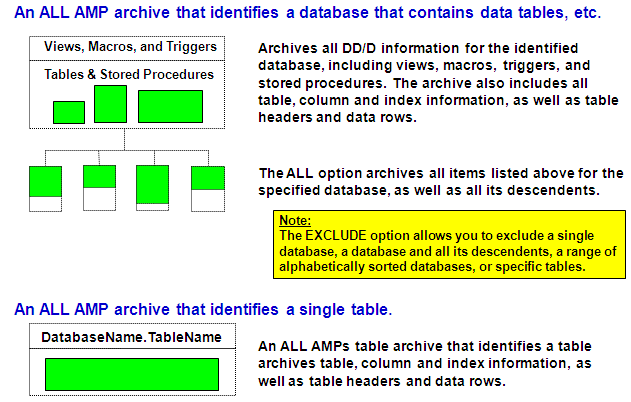
**This script archives DBC and all of its child databases/users.**

**Types of Archives :-**

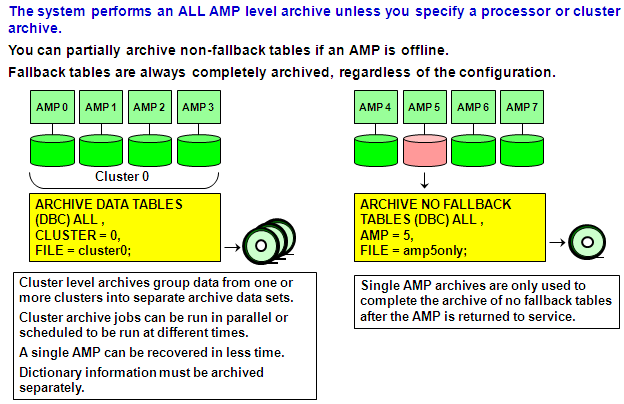




**Archive Objects :-**



**Archive Levels :-**



**Archive Options :-**



Summary

* Archive and Recovery (ARC) is a command-line utility that performs three operations: archive, restore and recovery.
* Archive and recovery jobs operate in two phases: dictionary and data phase.
* 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.
* The four types of archives are all-AMP database archive, all-AMP-table archive, specific-AMP or cluster archive and dictionary archive.

**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.

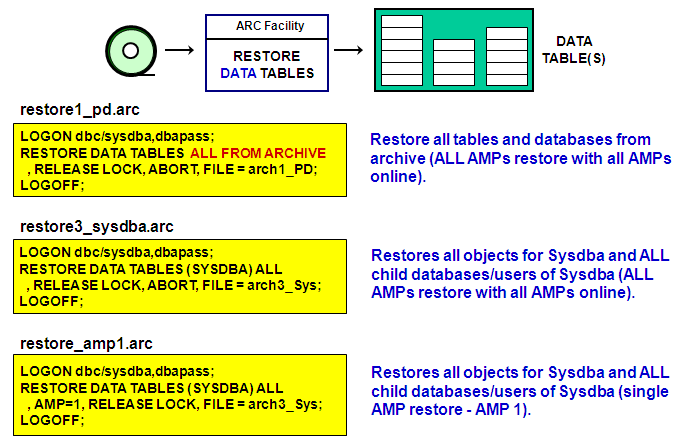
**Notes:**

* + 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.

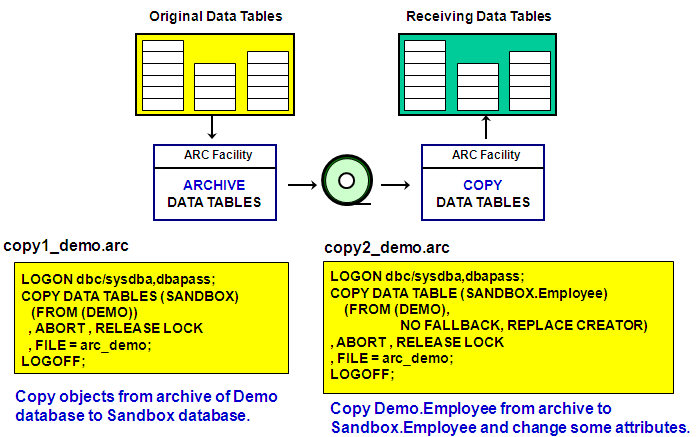
**Restore-Related Statements :-**

* 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 Validates inconsistent restraints against a target table thereby
* REFERENCES allowing users to execute UPDATE, INSERT and DELETE statements on the tables.

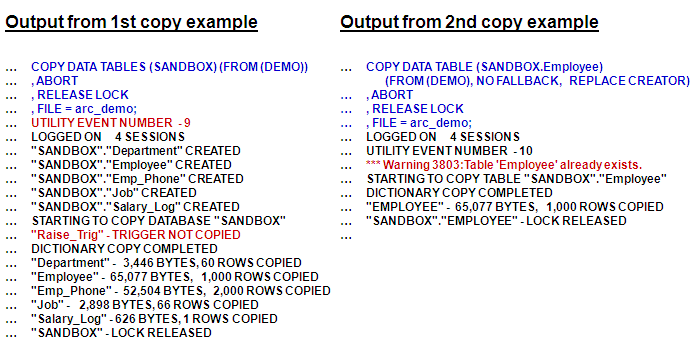
**RESTORE Examples :-**



**Copying Objects :-**



**Output of Copying Objects :-**



**Summary :-**

* Restore operations transfer database information from archive files stored on portable media to all AMP vprocs, AMP clusters or specified AMP vprocs.
* Archive and Restore (ARC) is a command-line utility you can use to restore data.
* 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.