Teradata Performance Optimization & Tuning Guide: The 100+ SQL Arsenal

This guide provides a comprehensive SQL-driven framework for assessing, monitoring, and improving Teradata system performance.

Critical Note: All "Action" queries in this guide are designed as **SQL Generators**. They query the Data Dictionary (DBC) to produce the executable SQL commands you need, avoiding the need for you to manually type database and table names.

1. System Assessment: Health Checks & Configuration

Assessment focuses on the static state of the system: data distribution, object definitions, and statistics health.

A. Storage & Skew Analysis (Access & Perm)

Skew is the primary performance killer. These queries identify where data is unbalanced across the entire system.

```
/* 1. Database level space usage summary */
SELECT DatabaseName, SUM(CurrentPerm)/1024**3 as GB_Used, SUM(MaxPerm)/1024**3 as
GB_Max, (GB_Used/NULLIF(GB_Max,0))*100 as Pct_Used FROM DBC.DiskSpace GROUP BY 1
ORDER BY 2 DESC;

/* 2. Identify tables consuming the most space (Top 50 System-wide) */
SELECT DatabaseName, TableName, SUM(CurrentPerm)/1024**3 AS GB_Size FROM
DBC.TableSize GROUP BY 1,2 ORDER BY 3 DESC TOP 50;

/* 3. Calculate Table Skew Factor for the Top 20 Largest Tables */
SELECT TOP 20 DatabaseName, TableName, SUM(CurrentPerm) AS TotalSize,
(MAX(CurrentPerm) - AVG(CurrentPerm)) / NULLIF(MAX(CurrentPerm), 0) * 100 AS
SkewFactor FROM DBC.TableSize GROUP BY 1,2 ORDER BY TotalSize DESC;

/* 4. Check for Empty Tables (Housekeeping) */
SELECT DatabaseName, TableName FROM DBC.TableSize GROUP BY 1,2 HAVING
SUM(CurrentPerm) = 0;

/* 5. Check Spool Space allocation per user */
SELECT DatabaseName, SpoolSpace/1024**3 AS Spool_GB FROM DBC.Databases WHERE
SpoolSpace > 0 ORDER BY 2 DESC;
```

```
/* 6. Identify databases with 0 Perm Space left */
SELECT DatabaseName, SUM(MaxPerm) - SUM(CurrentPerm) AS FreeSpace FROM
DBC.DiskSpace GROUP BY 1 HAVING FreeSpace <= 0;

/* 7. Identify the specific AMPs holding the most data (Hot AMPs) */
SELECT Vproc, SUM(CurrentPerm)/1024**3 AS GB_On_Amp FROM DBC.DiskSpace GROUP BY
1 ORDER BY 2 DESC;

/* 8. Check Temporary Space usage by Database */
SELECT DatabaseName, SUM(CurrentTemp) FROM DBC.DiskSpace GROUP BY 1 HAVING
SUM(CurrentTemp) > 0;

/* 9. Identify largest tables (Top 50) missing Fallback protection */
SELECT TOP 50 T.DatabaseName, T.TableName, S.SizeGB FROM DBC.Tables T JOIN (SELECT
DatabaseName, TableName, SUM(CurrentPerm)/1024**3 as SizeGB FROM DBC.TableSize
GROUP BY 1,2) S ON T.DatabaseName = S.DatabaseName AND T.TableName = S.TableName
WHERE T.ProtectionType = 'N' ORDER BY 3 DESC;

/* 10. Generate SQL to check granular distribution for the #1 most skewed table */
SELECT TOP 1 'SELECT Vproc, CurrentPerm FROM DBC.TableSize WHERE DatabaseName = '''
|| DatabaseName || ''' AND TableName = ''' || TableName || ''' ORDER BY 2 DESC;' as
Auto_Gen_SQL FROM DBC.TableSize GROUP BY 1,2 HAVING (MAX(CurrentPerm) -
AVG(CurrentPerm)) > 0 ORDER BY SUM(CurrentPerm) DESC;
```

B. Index Efficiency & Physical Design

Bad Primary Indexes (PI) cause skew and distribution collisions.

```
/* 11. Find tables with NO Primary Index (NoPI tables - exclude staging) */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE TableKind = 'T' AND
PrimaryKeyIndexId IS NULL;

/* 12. List tables with Duplicate Row checks (SET tables) */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE CheckOpt = 'Y';

/* 13. Identify Secondary Indexes (USI/NUSI) on large tables */
SELECT DatabaseName, TableName, IndexName, IndexType FROM DBC.Indices WHERE
IndexType IN ('S', 'U') ORDER BY DatabaseName, TableName;

/* 14. Check for Join Indexes (JI) */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE TableKind = 'I';
```

```
/* 15. Generate SQL to check Hash Collisions for the largest table in the system */
/* Note: Assumes first column of PI for simplicity in generation */
SELECT TOP 1 'SELECT HASHROW(' || ColumnName || '), COUNT(*) FROM ' || DatabaseName ||
'.' || TableName || ' GROUP BY 1 HAVING COUNT(*) > 1000 ORDER BY 2 DESC;' AS
Auto_Gen_SQL FROM DBC.IndicesV WHERE IndexType = 'P' AND ColumnPosition = 1 ORDER
BY (SELECT SUM(CurrentPerm) FROM DBC.TableSize WHERE DatabaseName =
DBC.IndicesV.DatabaseName AND TableName = DBC.IndicesV.TableName) DESC;

/* 16. Identify Partitioned Tables (PPI) */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE PartitioningColumn IS NOT
NULL;

/* 17. Find potential "Over-Indexed" tables (More than 5 Indexes) */
SELECT DatabaseName, TableName, COUNT(*) as IndexCount FROM DBC.Indices GROUP BY
1,2 HAVING IndexCount > 5;

/* 18. Check for Value-Ordered NUSIs */
SELECT DatabaseName, TableName, IndexName FROM DBC.Indices WHERE IndexType = 'V';

/* 19. Identify tables with BLOB/CLOB columns */
SELECT DatabaseName, TableName, ColumnName FROM DBC.Columns WHERE ColumnType
IN ('BO', 'CO');

/* 20. Find Tables with Column Partitioning (CP) */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE PartitioningColumn LIKE
'%COLUMN%';
```

C. Statistics Health (The Optimizer's Brain)

Missing stats cause "Product Joins" and terrible execution plans.

```
/* 21. Tables with NO statistics collected ever */
SELECT DatabaseName, TableName FROM DBC.Tables T WHERE NOT EXISTS (SELECT 1
FROM DBC.StatsV S WHERE T.DatabaseName = S.DatabaseName AND T.TableName =
S.TableName);

/* 22. Identify Stale Statistics (Older than 30 days) */
SELECT DatabaseName, TableName, ColumnName, LastCollectTimeStamp FROM DBC.StatsV
WHERE LastCollectTimeStamp < CURRENT_DATE - 30;

/* 23. Check Stats on Partition Columns (Critical for PPI performance) */
SELECT S.DatabaseName, S.TableName, S.ColumnName FROM DBC.StatsV S JOIN
```

```
DBC.Tables T ON S.DatabaseName = T.DatabaseName AND S.TableName = T.TableName
WHERE T.PartitioningColumn IS NOT NULL;

/* 24. Find Stats with 0 RowCount (Empty stats) */
SELECT DatabaseName, TableName, ColumnName FROM DBC.StatsV WHERE RowCount = 0;

/* 25. Check for Sampled Stats (Percentage < 100) */
SELECT DatabaseName, TableName, ColumnName, SampleSizePct FROM DBC.StatsV WHERE
SampleSizePct < 100;

/* 26. Identify Summary Statistics (Table level) */
SELECT DatabaseName, TableName, LastCollectTimeStamp FROM DBC.StatsV WHERE
ColumnName IS NULL;

/* 27. Check histogram details (Max intervals) */
SELECT DatabaseName, TableName, ColumnName, NumIntervals FROM DBC.StatsV ORDER
BY NumIntervals DESC;

/* 28. Identify stats collected on non-indexed columns */
SELECT S.DatabaseName, S.TableName, S.ColumnName FROM DBC.StatsV S LEFT JOIN
DBC.Indices I ON S.DatabaseName = I.DatabaseName AND S.TableName = I.TableName AND
S.ColumnName = I.ColumnName WHERE I.ColumnName IS NULL;

/* 29. Generate "HELP STATISTICS" command for the top 10 largest tables */
SELECT TOP 10 'HELP STATISTICS ' || DatabaseName || '.' || TableName || ';' AS Auto_Gen_SQL
FROM DBC.TableSize GROUP BY 1,2 ORDER BY SUM(CurrentPerm) DESC;

/* 30. Check stats collection duration (Long running collects) */
SELECT QueryText, (FirstRespTime - StartTime) DAY(4) TO SECOND FROM DBC.DBQLogTbl
WHERE QueryText LIKE 'COLLECT STAT%' ORDER BY 2 DESC;
```

2. Performance Monitoring: Real-Time & Forensics

Monitoring is split into "What is happening NOW" and "What happened THEN".

A. Real-Time Session Monitoring

Use these when the phone rings and users say "The system is slow."

```
/* 31. Basic "Who is logged on?" */
SELECT UserName, SessionNo, LogonTime FROM DBC.SessionInfoV;

/* 32. Active Queries consuming CPU right now */
```

```
SELECT SessionNo, UserName, CPUUsage, SqlTextInfo FROM DBC.SessionInfoV WHERE
AMPState = 'ACTIVE' ORDER BY CPUUsage DESC;

/* 33. Active Queries with high I/O skew (Hot AMP) */
SELECT SessionNo, UserName, AMPState, CurrentAMP, IOUsage FROM DBC.SessionInfoV
WHERE AMPState = 'ACTIVE' ORDER BY IOUsage DESC;

/* 34. Check for "Parse" state (Optimizer stuck) */
SELECT SessionNo, UserName, PEState FROM DBC.SessionInfoV WHERE PEState = 'PARSING';

/* 35. Identify blocked sessions (Waiters) */
SELECT SessionNo, UserName, AMPState FROM DBC.SessionInfoV WHERE AMPState =
'BLOCKED';

/* 36. Find sessions with high Spool Usage */
SELECT SessionNo, UserName, SpoolUsage FROM DBC.SessionInfoV ORDER BY SpoolUsage
DESC;

/* 37. Identify source IP of active sessions */
SELECT SessionNo, UserName, LogonSource FROM DBC.SessionInfoV;

/* 38. Check Request Priority (Workload Management) */
SELECT SessionNo, UserName, AccountName, Priority FROM DBC.SessionInfoV;

/* 39. Find long-running idle sessions */
SELECT SessionNo, UserName, LogonTime, LastRespTime FROM DBC.SessionInfoV WHERE
AMPState = 'IDLE' AND LastRespTime < CURRENT_TIMESTAMP - INTERVAL '4' HOUR;

/* 40. Count active sessions per user */
SELECT UserName, COUNT(*) FROM DBC.SessionInfoV GROUP BY 1 ORDER BY 2 DESC;
```

B. Locking & Blocking Analysis

Lock contention kills concurrency.

```
/* 41. Who is blocking whom? (Standard) */
SELECT Blk1UserId, Blk1SessionNo, Blk2UserId, Blk2SessionNo, BlockType FROM DBC.LockTbl;

/* 42. Count of locks by database */
SELECT DatabaseName, COUNT(*) FROM DBC.Locks GROUP BY 1;

/* 43. Identify Write Locks (Exclusive) */
```

```
SELECT SessionNo, DatabaseName, TableName, Mode FROM DBC.Locks WHERE Mode =
'WRITE';

/* 44. Identify Exclusive Locks */
SELECT SessionNo, DatabaseName, TableName, Mode FROM DBC.Locks WHERE Mode =
'EXCLUSIVE';

/* 45. Check Transaction Transaction Locks */
SELECT SessionNo, DatabaseName, TableName FROM DBC.Locks WHERE RequestType =
'TRANSACTION';

/* 46. Find Deadlocks (from history) */
SELECT * FROM DBC.DBQLogTbl WHERE ErrorCode = '2631';

/* 47. Check Lock delays in DBQL */
SELECT QueryID, LockDelay, QueryText FROM DBC.DBQLogTbl WHERE LockDelay > 0 ORDER
BY LockDelay DESC;

/* 48. Locks on System Tables (Dangerous) */
SELECT * FROM DBC.Locks WHERE DatabaseName = 'DBC';

/* 49. Sessions waiting for a lock */
SELECT SessionNo, Identify, TableName FROM DBC.ResLocks WHERE LockState = 'WAITING';

/* 50. Global Lock Manager status (Stub for GLM monitoring) */
SELECT 'Check Viewpoint for GLM locks' as Status;
```

C. Historical Analysis (DBQL)

The Database Query Log is the black box flight recorder.

```
/* 51. Top 10 CPU Consumers yesterday */
SELECT TOP 10 QueryID, UserName, TotalCPUTime, QueryText FROM DBC.DBQLogTbl
WHERE LogDate = DATE-1 ORDER BY TotalCPUTime DESC;

/* 52. Top 10 I/O Consumers yesterday */
SELECT TOP 10 QueryID, UserName, TotalIOCount, QueryText FROM DBC.DBQLogTbl WHERE
LogDate = DATE-1 ORDER BY TotalIOCount DESC;

/* 53. Queries with High CPU Skew (>90%) */
SELECT QueryID, AmpCpuTime, TotalCPUTime,
(1-(AmpCpuTime/(TotalCPUTime/NULLIF(NumOfActiveAmps,0))))*100 as Skew FROM
```

```
DBC.DBQLogTbl WHERE LogDate=DATE-1 AND TotalCPUTime > 1000 ORDER BY Skew DESC;

/* 54. Find "All-AMPs" operations that returned few rows */
SELECT QueryID, TotalIOCount, NumResultRows FROM DBC.DBQLogTbl WHERE
LogDate=DATE-1 AND TotalIOCount > 100000 AND NumResultRows < 100;

/* 55. Identify Product Joins (Look for huge CPU/IO ratio) */
SELECT QueryID, TotalCPUTime, TotalIOCount, (TotalCPUTime/NULLIF(TotalIOCount,0)) as
Ratio FROM DBC.DBQLogTbl WHERE Ratio > 10 AND LogDate=DATE-1;

/* 56. Queries that failed with "No More Spool Space" */
SELECT QueryID, UserName, ErrorCode, QueryText FROM DBC.DBQLogTbl WHERE ErrorCode
= 2646;

/* 57. Queries utilizing Fallback (Indication of AMP down or specific settings) */
SELECT QueryID, QueryText FROM DBC.DBQLogTbl WHERE ReqPhysIOKB > 0;

/* 58. Identify frequent small queries (Tactical workload) */
SELECT UserName, COUNT(*) FROM DBC.DBQLogTbl WHERE TotalCPUTime < 1 AND
LogDate=DATE-1 GROUP BY 1;

/* 59. Average execution time per hour */
SELECT EXTRACT(HOUR FROM StartTime), AVG(TotalCPUTime) FROM DBC.DBQLogTbl
WHERE LogDate=DATE-1 GROUP BY 1;

/* 60. Find queries performing Full Table Scans (heuristic) */
SELECT QueryID, QueryText FROM DBC.DBQLogTbl WHERE QueryText LIKE '%ALL-AMPs%'
AND QueryText NOT LIKE '%PRIMARY KEY%';

/* 61. Check Utility Usage (FastLoad/MultiLoad) */
SELECT AppID, COUNT(*) FROM DBC.DBQLogTbl WHERE LogDate=DATE-1 GROUP BY 1;

/* 62. Queries with longest Parser Duration */
SELECT QueryID, (FirstStepTime - StartTime) as ParseTime FROM DBC.DBQLogTbl ORDER BY
ParseTime DESC;

/* 63. Cache Hit Ratio estimates */
SELECT QueryID, ReqPhysIO, ReqIOKB FROM DBC.DBQLogTbl WHERE LogDate=DATE-1;

/* 64. Identify complex queries (High number of steps) */
SELECT QueryID, NumSteps FROM DBC.DBQLogTbl ORDER BY NumSteps DESC;

/* 65. Check for unwanted Cartesian Products */
```

```
SELECT QueryID, QueryText FROM DBC.DBQLogTbl WHERE QueryText LIKE '%CROSS
JOIN%';
```

D. Resource Usage & AMP Worker Tasks (AWT)

Deep dive into node/AMP health.

```
/* 66. Check CPU Utilization per Node */
SELECT TheDate, TheTime, NodeID, CPUIdle, CPUIOWait, CPUUServ, CPUUExec FROM
DBC.ResUsageSPMA WHERE TheDate=DATE-1;

/* 67. Identify Flow Control (AWT Exhaustion) */
SELECT TheDate, TheTime, NodeID, FlowControlled FROM DBC.ResUsageSPMA WHERE
FlowControlled > 0;

/* 68. Max AWT usage per AMP */
SELECT TheDate, TheTime, NodeID, Vproc1, MaxWorkMsg0 FROM DBC.ResUsageSAWT;

/* 69. Check Swap paging rates */
SELECT TheDate, TheTime, NodeID, PageMajorFaults FROM DBC.ResUsageSPMA;

/* 70. Monitor Host Channel activity */
SELECT TheDate, TheTime, NodeID, NetTxKB, NetRxKB FROM DBC.ResUsageSPMA;
```

3. Performance Improvement: Tactical Tuning

Improving performance involves DDL changes, improved SQL patterns, and better maintenance.

Note: The DDL examples here use VOLATILE TABLES or **Dynamic SQL Generation** to ensure they are runnable without external database dependencies.

A. Indexing & DDL Strategies

```
/* 71. Create Unique Primary Index (Volatile Example) */
CREATE VOLATILE TABLE Perf_Test_UPI (ID INT, Name VARCHAR(50)) UNIQUE PRIMARY INDEX
(ID) ON COMMIT PRESERVE ROWS;

/* 72. Create Non-Unique Primary Index (Volatile Example) */
CREATE VOLATILE TABLE Perf_Test_NUPI (EventDate DATE, Msg VARCHAR(100)) PRIMARY
INDEX (EventDate) ON COMMIT PRESERVE ROWS;
```

```
/* 73. Add Unique Secondary Index (USI) - Example Setup */
CREATE VOLATILE TABLE Perf_Test_USI (ID INT, Email VARCHAR(100)) PRIMARY INDEX (ID) ON
COMMIT PRESERVE ROWS;
CREATE UNIQUE INDEX (Email) ON Perf_Test_USI;

/* 74. Add Non-Unique Secondary Index (NUSI) - Example Setup */
CREATE VOLATILE TABLE Perf_Test_NUSI (ID INT, DeptID INT) PRIMARY INDEX (ID) ON
COMMIT PRESERVE ROWS;
CREATE INDEX (DeptID) ON Perf_Test_NUSI;

/* 75. Create Value-Ordered NUSI (Great for date ranges) */
CREATE VOLATILE TABLE Perf_Test_VO (ID INT, OrderDate DATE) PRIMARY INDEX (ID) ON
COMMIT PRESERVE ROWS;
CREATE INDEX (OrderDate) ORDER BY VALUES ON Perf_Test_VO;

/* 76. Create Partitioned Primary Index (PPI) by Date */
CREATE VOLATILE TABLE Perf_Test_PPI (ID INT, SaleDate DATE) PRIMARY INDEX(ID)
PARTITION BY RANGE_N(SaleDate BETWEEN DATE '2020-01-01' AND DATE '2025-12-31'
EACH INTERVAL '1' MONTH) ON COMMIT PRESERVE ROWS;

/* 77. Create Multi-Level PPI */
CREATE VOLATILE TABLE Perf_Test_MLPPI (ID INT, Region INT, SaleDate DATE) PRIMARY
INDEX(ID) PARTITION BY (RANGE_N(SaleDate BETWEEN DATE '2020-01-01' AND DATE
'2025-12-31' EACH INTERVAL '1' MONTH), CASE_N(Region=1, Region=2, NO CASE)) ON
COMMIT PRESERVE ROWS;

/* 78. Generate ALTER TABLE FALLBACK commands for tables missing it */
SELECT 'ALTER TABLE ' || DatabaseName || '.' || TableName || ', FALLBACK;' AS Auto_Gen_SQL
FROM DBC.Tables WHERE ProtectionType = 'N' AND TableKind = 'T';

/* 79. Generate DROP INDEX commands for secondary indexes that are rarely used (Simulated
logic) */
/* Requires DBQL access usage analysis, here we simply list all USI/NUSI for review */
SELECT 'DROP INDEX ' || IndexName || ' ON ' || DatabaseName || '.' || TableName || ';' AS
Auto_Gen_SQL FROM DBC.Indices WHERE IndexType IN ('S','U');

/* 80. Rename table logic (Volatile Example) */
CREATE VOLATILE TABLE Prod_Table (ID INT) ON COMMIT PRESERVE ROWS;
RENAME TABLE Prod_Table TO Old_Table;
```

B. Statistics Collection Generators

Instead of guessing table names, use these queries to **generate** the commands you need for your environment.

```
/* 81. Generate COLLECT STATS on Primary Index for all tables in a Database */
/* Note: Replace 'YourDB' with your actual database name when running the result */
SELECT 'COLLECT STATISTICS COLUMN(' || ColumnName || ') ON ' || DatabaseName || '.' ||
TableName || ';' AS Auto_Gen_SQL FROM DBC.Indices WHERE IndexType = 'P';

/* 82. Generate COLLECT STATS on Partition Columns */
SELECT 'COLLECT STATISTICS COLUMN(PARTITION) ON ' || DatabaseName || '.' || TableName
|| ';' AS Auto_Gen_SQL FROM DBC.Tables WHERE PartitioningColumn IS NOT NULL;

/* 83. Generate RE-COLLECT STATS for tables with stale stats (>30 days) */
SELECT 'COLLECT STATISTICS ON ' || DatabaseName || '.' || TableName || ';' AS Auto_Gen_SQL
FROM DBC.StatsV WHERE LastCollectTimeStamp < CURRENT_DATE - 30 GROUP BY
DatabaseName, TableName;

/* 84. Generate COLLECT STATS for Empty Statistics (0 rows) */
SELECT 'COLLECT STATISTICS ON ' || DatabaseName || '.' || TableName || ';' AS Auto_Gen_SQL
FROM DBC.StatsV WHERE RowCount = 0 GROUP BY DatabaseName, TableName;

/* 85. Generate COPY STATISTICS commands */
/* Generates a template to copy stats from a Prod DB to a Dev DB */
SELECT 'COLLECT STATISTICS FROM ' || DatabaseName || '.' || TableName || ' TO DevDB.' ||
TableName || ';' AS Auto_Gen_SQL FROM DBC.Tables;

/* 86. View Help Stats (Dynamic) */
SELECT TOP 1 'HELP STATISTICS ' || DatabaseName || '.' || TableName || ';' FROM DBC.Tables;

/* 87. Show detailed histogram data (Dynamic) */
SELECT TOP 1 'SHOW STATISTICS VALUES COLUMN(' || ColumnName || ') ON ' ||
DatabaseName || '.' || TableName || ';' FROM DBC.StatsV;

/* 88. Generate DROP STATISTICS for a specific column */
SELECT TOP 1 'DROP STATISTICS COLUMN(' || ColumnName || ') ON ' || DatabaseName || '.' ||
TableName || ';' FROM DBC.StatsV;

/* 89. Generate RE-COLLECT all defined stats */
SELECT 'COLLECT STATISTICS ON ' || DatabaseName || '.' || TableName || ';' AS Auto_Gen_SQL
FROM DBC.Tables;

/* 90. Diagnostic help stats (Simulation) */
SELECT TOP 1 'DIAGNOSTIC HELP STATISTICS ON ' || DatabaseName || '.' || TableName || ';'
```

```
FROM DBC.Tables;
```

C. Query Optimization Techniques

Self-contained examples using Volatile Tables.

```
/* 91. EXPLAIN a query (The starting point) */
CREATE VOLATILE TABLE Perf_Explain_Demo (ID INT) ON COMMIT PRESERVE ROWS;
EXPLAIN SELECT * FROM Perf_Explain_Demo;

/* 92. Use DISTINCT to remove duplicates */
SELECT DISTINCT DatabaseName FROM DBC.Tables;

/* 93. Use GROUP BY to aggregate (Often faster than DISTINCT) */
SELECT DatabaseName FROM DBC.Tables GROUP BY 1;

/* 94. Top N optimization */
SELECT TOP 10 * FROM DBC.Tables;

/* 95. Sample data for quick checks */
SELECT * FROM DBC.Tables SAMPLE 100;

/* 96. Use LOCKING ROW FOR ACCESS (Dirty Read - Improves concurrency) */
LOCKING ROW FOR ACCESS SELECT * FROM DBC.Tables;

/* 97. Volatile Table Creation (For intermediate steps) */
CREATE VOLATILE TABLE VT_Temp AS (SELECT DatabaseName, TableName FROM
DBC.Tables) WITH DATA ON COMMIT PRESERVE ROWS;

/* 98. Global Temporary Table (DDL Example) */
/* GTTs persist across sessions, unlike Volatile tables */
CREATE GLOBAL TEMPORARY TABLE GTT_Temp (ID INT) ON COMMIT PRESERVE ROWS;

/* 99. MERGE INTO (Upsert optimization) */
CREATE VOLATILE TABLE Target_T (ID INT, Val VARCHAR(10)) ON COMMIT PRESERVE ROWS;
CREATE VOLATILE TABLE Source_S (ID INT, Val VARCHAR(10)) ON COMMIT PRESERVE ROWS;
MERGE INTO Target_T T USING Source_S S ON T.ID = S.ID WHEN MATCHED THEN UPDATE
SET Val = S.Val WHEN NOT MATCHED THEN INSERT (S.ID, S.Val);

/* 100. Insert Explain into Table (For saving plans) */
CREATE VOLATILE TABLE MyPlanTable AS (SELECT * FROM DBC.QryLogExplainV) WITH NO
DATA ON COMMIT PRESERVE ROWS;
```

```
INSERT EXPLAIN INTO MyPlanTable SELECT * FROM DBC.Tables;
```

D. Advanced Administration & Cleanup

```
/* 101. Check Multi-Value Compression (MVC) Candidates (Dynamic Generation) */
SELECT TOP 10 'SHOW TABLE ' || DatabaseName || '.' || TableName || ';' AS Auto_Gen_SQL
FROM DBC.TableSize ORDER BY CurrentPerm DESC;

/* 102. Generate ALTER TABLE to add compression (Template) */
SELECT TOP 1 'ALTER TABLE ' || DatabaseName || '.' || TableName || ' ADD ' || ColumnName || '
COMPRESS;' AS Auto_Gen_SQL FROM DBC.Columns;

/* 103. Release Lock manually (Careful!) */
SELECT TOP 1 'RELEASE LOCK ' || DatabaseName || '.' || TableName || ', OVERRIDE;' AS
Auto_Gen_SQL FROM DBC.Locks;

/* 104. Abort a specific session (Template) */
SELECT 'CALL DBC.AbortSession(' || HostId || ', ' || SessionNo || ', ''Y'', ''Performance Abort'');'
AS Auto_Gen_SQL FROM DBC.SessionInfoV WHERE AMPState = 'ACTIVE';

/* 105. Check for tables requiring reorganization */
/* No direct SQL, requires reading Show Table or output of CheckTable utility */
SELECT 'Run CheckTable on System' as Recommendation;

/* 106. Find Macros */
SELECT DatabaseName, TableName FROM DBC.Macros;

/* 107. Find Stored Procedures */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE TableKind = 'P';

/* 108. Find Triggers */
SELECT DatabaseName, TableName FROM DBC.Triggers;

/* 109. Check Role Memberships */
SELECT * FROM DBC.RoleMembers;

/* 110. List all users with a specific role (Dynamic) */
SELECT UserName FROM DBC.RoleMembers WHERE RoleName IN (SELECT RoleName FROM
DBC.Roles);

/* 111. Identify Views */
SELECT DatabaseName, TableName FROM DBC.Tables WHERE TableKind = 'V';
```

```
/* 112. Show View Definition (Dynamic) */
SELECT TOP 1 'SHOW VIEW ' || DatabaseName || '.' || TableName || ';' FROM DBC.Tables WHERE
TableKind = 'V';

/* 113. Create a View for security (Volatile Example) */
CREATE VOLATILE TABLE Secure_Table (ID INT, Secret VARCHAR(100)) ON COMMIT
PRESERVE ROWS;
CREATE VIEW Public_View AS LOCKING ROW FOR ACCESS SELECT ID FROM Secure_Table;

/* 114. Check current date/time settings */
SELECT CURRENT_DATE, CURRENT_TIMESTAMP, CURRENT_USER;

/* 115. Flush Query Log (Force write to disk) */
FLUSH QUERY LOG;
```