

Implementing Performance in Reentrant Layered Logic Tables (RLLT)

Colin James III, Principal Scientist, CEC Services, LLC, 1613 Morning Dr, Loveland CO 80538-4410, iprllt@cec-services.com, C: 719.210.9534 , F: 970.593.1350

Keywords

Logic Table Technology, LTT, Layered Logic Tables, LLT, Reentrant Layered Logic Tables, RLTT, process automation, real time performance, scheduling, SQL

Abstract

In the programming of process automation, the rules for dispatching such processes are mapped directly into the rows and columns of a logic table in a relational database. Layers of logic tables apply to the more complex processing instructions for water treatment. For reentrant logic tables of eight rows and columns at eight layers, the possible complex processes number over 2^122. Metrics on desktop computers show that Reentrant Layered Logic Tables (RLLT) perform fast enough in real time to process eight layers of such tables in 0.07 seconds. To increase performance, the logic tables are implemented as columns of the data type of variable character in the same logic table. In structured query language (SQL), the generic access form of the layered logic tables is [AND EXISTS] (SELECT … FROM … WHERE SUBSTR … = [valid switch]).

Reentrant Layered Logic Tables

A logic table contains switches to instruct a process controller as to which process to invoke per logic sequence position, as in Table 1 below.

The hierarchy of layered logic tables below follow the order of six subsets derived from the requirements of James 2002.5 in the order of most specific to most abstract in tables below as: 1. Procedures; 2. Processes; 3. Tasks; 4. Units; 5. Components; and 6. Requirements.

Logic Table 1 for Processes:

Row	**Column Position Number**		
Number	Process 1	Process 2	Process M
Procedure 1	Process Switches		
Procedure 2			
Procedure N			

Logic Table 2 for Tasks:

Row	**Column Position Number**		
Number	Task 1	Task 2	Task M
Process 1	Task Switches		
Process 2			
Process N			

Logic Table 3 for Units:

Row	**Column Position Number**		
Number	Unit 1	Unit 2	Unit M
Task 1	Unit Switches		
Task 2			
Task N			

Logic Table 4 for Components:

Row	**Column Position Number**		
Number	Comp1	Comp2	Comp M
Units 1	Component Switches		
Units 2			
Units N			

Logic Table 5 for Requirements:

Row	**Column Position Number**		
Number	Reqt 1	Reqt 2	Reqt M
Comp 1	Requirement Switches		
Comp 2			
Comp N			

Generic SQL Code for Layered Logic Tables

The generic SQL code for the layered logic tables 1-5 is below. The nested SQL query returns the procedures to be updated from the respective procedure switches.

```
SELECT   L5.procedure,   S5.switch
FROM     process AS L5, type_switch as S5
WHERE    SUBSTR( L5.switches, L4.process, 1) = S5.switch
AND EXISTS
     (SELECT  L4.process, S4.switch
     FROM     task as L4, type_switch as S4
     WHERE    SUBSTR( L4.switches, L3.task, 1) = S4.switch
     AND EXISTS
          (SELECT  L3.task, S3.switch
          FROM     unit as L3, type_switch as S3
          WHERE    SUBSTR( L3.switches, L2.unit, 1) = S3.switch
          AND EXISTS
               (SELECT  L2.unit, S2.switch
               FROM     component as L2, type_switch as S2
               WHERE    SUBSTR( L3.switches, L2.unit, 1) = S2.switch
               AND EXISTS
                    (SELECT  L1.component, S1.switch
                    FROM     requirement as L1, type_switch as S1
                    WHERE    SUBSTR( L1.switch, user_input_reqt, 1) = S1.switch ) ) ) )
```

Implementation of RLLT

The two main phases to implement performance into RLLT are designing the logic tables and setting the parameters of the relational database used.

The logic tables are implemented as columns of data type variable character. Because there are 64 such columns, the size of each is limited to 32 characters as VARCHAR(32). The columns are named with very short names such as sub01 … sub64. In the SQL access code, the column names for the logic tables are aliased as T1, T3, …, T127 in odd numbers. A single index column is reused and aliased as T1, T2, T3, …, T127 in sequential numbers.

For the queries to perform at all in real time due to the complexity of the nested levels, the parameters were set as follows for IBM DB2 7.

```
CONNECT TO LLT @
UPDATE DATABASE CONFIG FOR LLT USING stmtheap 60000 @
!! The statement heap must be significantly larger than the default 4096 [x 4 KB] @
UPDATE DB CFG FOR LLT USING app_ctl_heap_sz 1024 @
!! Note: the app_ctl_heap must be significantly larger than the default 64 [x 4 KB] @
SET CURRENT QUERY OPTIMIZATION 0 @
!! Note: the minimum optimization avoids elaborate access plans @

!! All of the values below are increased to either 1024 or 8192 @
UPDATE DB CFG FOR LLT USING dft_queryopt 0 @
UPDATE DB CFG FOR LLT USING  num_freqvalues 1024 @
UPDATE DB CFG FOR LLT USING  locklist 1024 @
UPDATE DB CFG FOR LLT USING  sortheap 8192 @
UPDATE DB CFG FOR LLT USING applheapsz 8192 @
UPDATE DB CFG FOR LLT USING  dbheap 8192 @
UPDATE DB CFG FOR LLT USING  catalogcache_sz 1024 @
UPDATE DB CFG FOR LLT USING  buffpage 8192 @

DISCONNECT all @
CONNECT TO LLT @
```

To record the performance, the level of iteration was suppressed.

```
!! DELETE FROM performance WHERE time_date IS NOT NULL @

!! For performance, the level number is suppressed by inserting 0 throughout @
!! To enable the storing of the number of iteration, this code is used: @
!! INSERT INTO performance (time_date, eta) VALUES (current timestamp, ( 1 +
(SELECT COUNT( *) from performance) / 2)) @
```

The format of the access code for each level is shown below for levels 1 through 3.

```
!! This begins level 1 @
INSERT INTO performance (time_date, eta) VALUES (current timestamp, 0) @
(SELECT T1.index FROM logic AS T1 WHERE  SUBSTR( T1.sub01, 1, 1) <> 'x')
@
INSERT INTO performance (time_date, eta) VALUES (current timestamp,
(SELECT ((current timestamp) - MAX( time_date)) from performance))@
DISCONNECT all @

!! This begins level 2 @
CONNECT TO LLT @
INSERT INTO performance (time_date, eta) VALUES (current timestamp, 0) @
(SELECT T3.index FROM logic AS T3, logic AS T2 WHERE SUBSTR( T3.sub02,
T2.index, 1) <> 'x' AND T2.index IN
(SELECT T1.index FROM logic AS T1 WHERE SUBSTR( T1.sub01, 1, 1) <> 'x'))
@
INSERT INTO performance (time_date, eta) VALUES (current timestamp,
(SELECT ((current timestamp) - MAX( time_date)) from performance))@
DISCONNECT all @

!! This begins level 3 @
CONNECT TO LLT @
INSERT INTO performance (time_date, eta) VALUES (current timestamp, 0) @
(SELECT T5.index FROM logic AS T5, logic AS T4 WHERE SUBSTR ( T5.sub03,
T4.index, 1) <> 'x' AND T4.index IN
(SELECT T3.index FROM logic AS T3, logic AS T2 WHERE SUBSTR( T3.sub02,
T2.index, 1) <> 'x' AND T2.index IN
(SELECT T1.index FROM logic AS T1 WHERE SUBSTR( T1.sub01, 1, 1) <>
'x'))) @
INSERT INTO performance (time_date, eta) VALUES (current timestamp,
(SELECT ((current timestamp) - MAX( time_date)) from performance))@
DISCONNECT all @
…
```

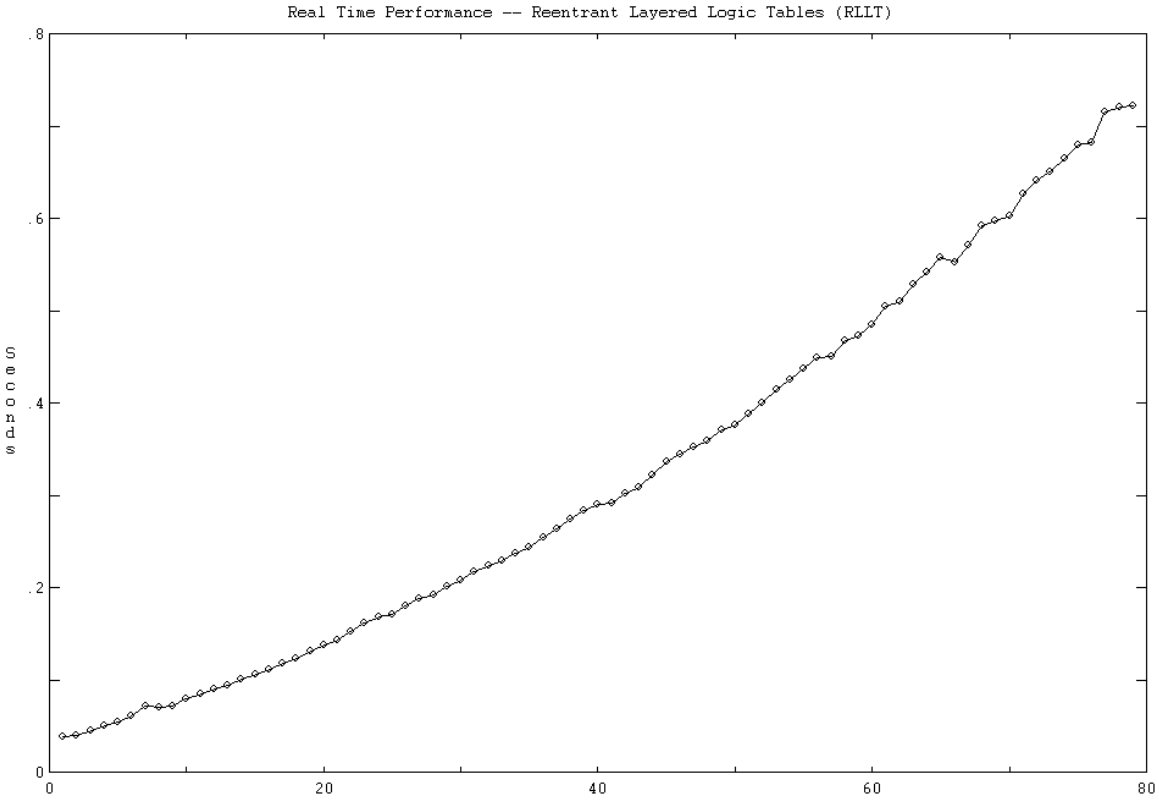
The routine for outputting the performance metrics is shown below after level 79 on page 136 in landscape format:

```
CONNECT TO LLT @
SELECT eta  FROM performance where eta <> 0.0 @
!! To print the level iteration use SELECT * FROM performance @
SELECT COUNT (*) FROM performance AS P2, performance AS P3 WHERE
P2.eta <> 0.0 AND P3.eta <> 0.0 AND P2.date_time < P3.date_time AND P2.eta >
P3.eta @
!! Checks for elapsed times not increasing in time ± 0.011 seconds for graphing @
```

Performance in Real Time

For a reentrant logic table of eight switches in seven rows, the number of possible switch combinations is 8! or 40320. For eight such tables with eight columns to be layered and reentrant, the number of possible switch combinations increases to (8!)^8 or about 2^122. By contrast for a reentrant logic table of 64 switches, 63 rows, and 64 columns for up to 64 levels of logic, the number of possible switch combinations is larger by a factor of about 2^154 at (64!)^64 or about 2^18943.

The graph below is mean time of 40 runs for a logic table of four switches in three rows, 79 columns, and up to 79 levels. For 64 columns and 64 levels, the number of possible switch combinations is therefore (4!)^64 or about 2^293.



The graph above shows that the real time performance for a logic table of 4 switches in 3 rows and 8 columns for 8 levels is 0.07 seconds from data in the program below. The number of possible switches is (4!)^8 or about 2^36.

The graph above shows that the real time performance for a logic table of 4 switches in 3 rows and 64 columns for 64 levels is 0.54 seconds. The number of possible switch combinations is again about 2^293. That time for the 64 levels is about 8 times greater than the time for the 8 levels. The number of possible switch combinations is over 2^8 or about 256 times greater than the number of possible switch combinations for the 8 levels. The graph above also shows that from 8 to 64 levels the performance is about linear.

The tests were performed on a commodity desktop computer with 733 MHz processor, 768 MB RAM, and 80 GB hard disk.

The anomalies that 8 and 9 levels are slightly faster than 7 levels and that 66 levels are slightly faster than 65 levels are due to the DB2 environment tested.

The data above is contained in the following computer program for graphs:

```
! SQL Graph - TrueBASIC® Source Code for graph
!
! Real time performance for 79-layers of logic tables
! Mean time for 40 runs
!
! © Copyright 2003 by CEC Services, LLC
! All Rights Reserved

LIBRARY "..\TBLibs\SGLib.trc"

DIM layers(79), secs(79)

MAT READ layers, secs

DATA 1,  2,  3,  4,  5,  6,  7,  8
DATA 9,  10, 11, 12, 13, 14, 15, 16
DATA 17, 18, 19, 20, 21, 22, 23, 24
DATA 25, 26, 27, 28, 29, 30, 31, 32
DATA 33, 34, 35, 36, 37, 38, 39, 40
DATA 41, 42, 43, 44, 45, 46, 47, 48
DATA 49, 50, 51, 52, 53, 54, 55, 56
DATA 57, 58, 59, 60, 61, 62, 63, 64
DATA 65, 66, 67, 68, 69, 70, 71, 72
DATA 73, 74, 75, 76, 77, 78, 79

DATA 0.038900, 0.040300, 0.045275, 0.050325, 0.053825, 0.060550, 0.071850, 0.069800
DATA 0.071100, 0.079600, 0.084375, 0.090200, 0.094200, 0.100875, 0.106400, 0.110950
DATA 0.118150, 0.122675, 0.131150, 0.137700, 0.142700, 0.152700, 0.161725, 0.167975
DATA 0.171250, 0.180725, 0.188600, 0.192300, 0.200725, 0.207550, 0.217550, 0.223600
DATA 0.229525, 0.236850, 0.244300, 0.254900, 0.264075, 0.273950, 0.283375, 0.290350
DATA 0.291650, 0.302300, 0.309200, 0.321875, 0.336250, 0.344450, 0.351700, 0.358300
DATA 0.370700, 0.376000, 0.387625, 0.399650, 0.415125, 0.424575, 0.436500, 0.448950
DATA 0.449775, 0.468175, 0.472425, 0.484175, 0.505150, 0.510150, 0.528175, 0.541400
DATA 0.558175, 0.552700, 0.570275, 0.592050, 0.596875, 0.602400, 0.626700, 0.640625
DATA 0.650700, 0.664900, 0.679750, 0.681750, 0.715000, 0.720275, 0.722000

CALL SetText ("Real Time Performance -- Reentrant Layered Logic Tables (RLLT)", "",
"Seconds")
CALL DataGraph (layers, secs, 4, 1, "black black")

END
```

What follows from the statistic above that the possible combinations in a square 64 table with 64 logic switches at about 2^18943 is that layered logic tables may be used for non invertive encryption as accessed directly by SQL engines.

Conclusion

RLLT is ideally suited for programming processes. The generalized implementation is:

```
[AND EXISTS]
  (SELECT  switches available, logic switches, items returned
  FROM     type of switches available, logic table of interest
  WHERE    SUBSTR( logic switches, input point, 1) = type of switches available)
```

Acknowledgments

Thanks are due to Larry Cagg of Cagg Enterprises for helpful discussions.

References

James, C. 2003.4, “Reentrant Layered Logic Tables (RLLT)”, [in submission to the Industrial Water Conference 2003 (IWC2003)], CEC Services, LLC, Loveland CO.

James, C. 2003.3, “Layered Logic Tables (LLT)”, unpublished, CEC Services, LLC, Loveland CO.

James, C. 2003.2, "Software Factory", unpublished brochure, CEC Services, LLC, Loveland CO.

James, C. 2003.1, "Software Factory", unpublished poster, CEC Services, LLC, Loveland CO.

James, C. 2002.5, "The Software Development Methodology [SDM]", unpublished, CEC Services, LLC, Loveland CO.

James, C. 2002.4, "Reentrant Logic Table Technology", unpublished, CEC Services, LLC, Loveland CO.

James, C. 2002.3, "Static and Dynamic Driver Triggers", unpublished, CEC Services, LLC, Loveland CO.

James, C. 2002.2, "Additional Information", unpublished, CEC Services, LLC, Loveland CO.

James, C. 2002.1, “Implementation Details for Multiple Billing”, CEC Services, LLC, Loveland CO.

James, C. 2001.2, "Report Accounts [RA] v 1.2 Inventory / Point of Sale", unpublished, CEC Services, LLC, Loveland CO.

James, C., 2001.1, "Report Accounts [RA] v 1.2", unpublished, CEC Services, LLC, Loveland CO.

James, C. 1999.1, “Recent Advances in Logic Tables for Reusable Database Engines”, Proceedings of the American Society of Mechanical Engineers International, Petroleum Division, 75th Anniversary Conference, Energy Sources Technology Conference & Exhibition, Houston, Texas.

James, C., 1998.5, "Multiple and Self-Modifying Logic Tables with Queries", unpublished, CEC Services, LLC, Loveland CO.

James, C. 1998.4, “A Reusable Database Engine for Accounting Arithmetic”, Proceedings of The Third Biennial World Conference on Integrated Design & Process Technology, Vol. 2, pp. 25-30, Berlin, Germany.

James, C., 1998.3, "Competency test for CEC Services, LLC", unpublished, CEC Services, LLC, Loveland CO.

James, C. 1998.2, “Theory and Application of Logic Tables in Relational Database Engines”, Doctoral Dissertation, Pacific Western University, Los Angeles.

James, C., 1998.1, "Ticket Reservations [TR] ver 1.1", unpublished, CEC Services, LLC, Loveland CO.

James, C., 1997.2, "User Documentation", unpublished, CEC Services, LLC, Loveland CO.

James, C., 1997.1, "Logic Table Design for Reports in RA", unpublished, CEC Services, LLC, Loveland CO.