

Layered Logic Tables (LLT)

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

Keywords

automatic programming, correlated sub query, dispatching, Layered Logic Tables, loom, LTT, manufacturing, scheduling, software factory automation, SQL, thread, weaving

Abstract

In the textile industry, the rules of dispatching thread for vertical and horizontal positions are mapped directly into rows and columns of a logic table in relational database. Layers of logic tables apply to loom instructions for weaving. In the software development industry, a software factory is built using the same principles and at the complexity of five levels. When abstracted, the generic form of layered logic tables in structured query language (SQL) is the code segment format of:

IN (SELECT … FROM … WHERE SUBSTR … = [valid switch]).

Introduction

In 1834 Charles Babbage attempted to program looms by his Analytical Engine. One of his students was Lady Ada Byron Lovelace, also known as the first female programmer. In her honor, the US Department of Defense named their programming language as Ada. Two hundred years later, the textile industry and the manufacturing sector continue as candidates for programming automation.

Layers of Logic Tables

A logic table contains switches to instruct a loom as to which horizontal color per vertical row to use per horizontal position, as in Table A below.

Logic Table A for Weaving:

Row Number	**Column Position Number** VThread 1	VThread 2
HThread 1	HColor 9	--
HThread 2	--	--
HThread 3	--	--
HThread 4	HColor 1	HColor 6

Logic Table A outputs in horizontal thread rows the respective horizontal thread color by vertical position. The input is vertical thread position and vertical thread color per task from Logic Table B below.

Logic Table B for Tasks of Weaving:

Row Number	**Column Position Number** Task 1	Task 2
VThread 1	VColor 3	--
VThread 2	VColor 4	--
VThread 3	--	VColor 2

From Table B, Task 1 designates the background vertical color 3 for vertical thread 1. From Table A: horizontal thread color 9 is for row 1; horizontal thread color 3 is for rows 2 through 3; and horizontal thread color 1 is for row 4. Task 1 also designates the background vertical color 4 for vertical thread 2. From Table A: horizontal thread color 4 is for rows 1 through 4; and horizontal thread color 6 is for row 4. Table B effectively serves as an index of layers of Table A.

A third Table C for requirements may be added for units of software requirements containing a series of tasks,

Logic Table C for Units of Tasks:

Row Number	**Column Position Number** Unit 1	Unit 2
Task 1	USwitch 2	--
Task 2	--	Uswitch 3

From Table C: Unit 1 designates software unit switch USwitch 2 for Task 1; and Unit 2 designates USwitch 3 for Task 3.

A Hierarchy of Layered Tables

The hierarchy of layered logic tables below follows the order of eight subsets derived from requirements in the order of most specific to most abstract: 1. Item; 2. Step; 3. Procedure; 4. Process; 5. Task; 6. Unit; 7. Component; and 8. Requirement. This hierarchy is compatible with the Software Development Methodology (SDM) [James 2002.5].

Logic Table 7 for Step:

Row Number	**Column Position Number** Step1	Step 2	Step M
Step Item N	Step Switches		

Logic Table 6 for Procedure:

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

Logic Table 5 for Process:

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

Logic Table 4 for Task:

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

Logic Table 3 for Unit:

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

Logic Table 2 for Component:

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

Logic Table 1 for Requirement:

Row Number	**Column Position Number** Requirement 1	Requirement 2	Requirement M
Requirement Component N	Requirement Switches		

Table design considerations

There are four basic designs to implement the logic tables above.

1. Each table contains two columns for row number and switches.

2. The tables are combined into one table where:

   2.1. One column is for row numbers, and one column of switches is for all logic tables concatenated together in one string.

   2.2. One column is for row numbers, and multiple columns of switches are for the respective logic tables.

   2.3. Multiple columns are for row numbers, and multiple columns of switches are for the respective logic tables.

For design 1, the positive feature is clarity of design with one logic table per level. The negative feature is using more logic tables because databases with more tables usually perform slower.

For design 2.1, the positive feature is using fewer logic tables. The negative feature is loss of clarity of design because the single string of logic switches are indexed using arbitrary constants unique to the string length of the run of each respective logic block.

For design 2.2, the positive features are direct indexing of any logic block as a column and the potential for the length or complexity of any logic block as a column to grow dynamically if the data type is a variable character VARCHAR(nnn). The negative feature is that in order for the column for row numbers to be reused many times for each set of switches, the column must be renamed for sub queries where the aliases are by table rather than by meaningful column names helpful for code maintenance.

For design 2.3, the positive feature is for columns for row numbers to be directly named for clarity without resorting to confusing aliases. The negative feature is more than one column for row numbers for each respective column of logic switches.

For clarity of implementation and subsequent code maintenance, design 2.3 was chosen.

Code implementation in DB2

For design 2.3, the implementation code below in SQL is specific to IBM DB2. The first section of code sets up the database.

```
DISCONNECT all @
!! In case the database already exists: DROP DATABASE LLT @
CREATE DATABASE LLT @
CONNECT TO LLT @
UPDATE DATABASE CONFIG FOR LLT USING stmtheap 32768 @
!! Note: the statement heap must be significantly more than the default 4096 @
SET CURRENT QUERY OPTIMIZATION 0 @
!! Note: the minimum optimization avoids elaborate access plans @

CREATE TABLE logic (
  reqt_comp          INTEGER NOT NULL,   reqt_switch       VARCHAR(254),
  comp_unit          INTEGER NOT NULL,   comp_switch       VARCHAR(254),
  unit_task          INTEGER NOT NULL,   unit_switch       VARCHAR(254),
  task_process       INTEGER NOT NULL,   task_switch       VARCHAR(254),
  process_procedure  INTEGER NOT NULL,   process_switch    VARCHAR(254),
  procedure_step     INTEGER NOT NULL,   procedure_switch  VARCHAR(254),
  step_item          INTEGER NOT NULL,   step_switch       VARCHAR(254))
  @
ALTER TABLE logic ADD PRIMARY KEY (reqt_comp) @

CREATE TABLE switch ( switch_id CHAR(1) NOT NULL) @
ALTER TABLE switch ADD PRIMARY KEY (switch_id) @

INSERT INTO logic ( reqt_comp, reqt_switch, comp_unit, comp_switch, unit_task,
        unit_switch, task_process, task_switch,  process_procedure, process_switch,
        procedure_step, procedure_switch, step_item, step_switch)
VALUES ( 1, '11x', 1, 'xx1', 1, '1xx', 1, 'xxx', 1, '1x1', 1, '1x1', 1, '1x1') @

INSERT INTO logic ( reqt_comp, reqt_switch, comp_unit, comp_switch, unit_task,
        unit_switch, task_process, task_switch,  process_procedure, process_switch,
        procedure_step, procedure_switch, step_item, step_switch)
VALUES ( 2, 'xxx', 2, 'xxx', 2, 'xxx', 2, '2x2', 2, 'x2x', 2, 'x2x', 2, 'x2x') @

INSERT INTO logic ( reqt_comp, reqt_switch, comp_unit, comp_switch, unit_task,
        unit_switch, task_process, task_switch,  process_procedure, process_switch,
        procedure_step, procedure_switch, step_item, step_switch)
VALUES ( 3, '3x3', 3, '3xx', 3, 'xx3', 3, '3xx', 3, 'xx3', 3, 'xx3', 3, 'xx3') @

INSERT INTO switch ( switch_id) VALUES ( '1') @
INSERT INTO switch ( switch_id) VALUES ( '2') @
INSERT INTO switch ( switch_id) VALUES ( '3') @
```

The second section of code specifies the main SELECT statement for the trigger to read the database. Some aliasing cannot be avoided, although it is still clear.

```
SELECT  L13.step_item
FROM    logic AS L13,  logic AS L12,  switch
WHERE   SUBSTR( L13.step_switch, L12.task_process, 1) = switch_id
AND     L12.task_process IN
    (SELECT  L11.procedure_step
    FROM     logic AS L11,  logic AS L10,  switch
    WHERE    SUBSTR( L11.procedure_switch, L10.process_procedure, 1) =
             switch_id
    AND     L10.task_process IN
        (SELECT  L9.process_procedure
        FROM     logic AS L9, logic AS L8, switch
        WHERE    SUBSTR( L9.process_switch, L8.task_process, 1) =
                 switch_id
         AND     L8.task_process IN
            (SELECT L7.task_process
            FROM     logic AS L7, logic AS L6, switch
            WHERE    SUBSTR ( L7.task_switch, L6.unit_task, 1) = switch_id
            AND      L6.unit_task IN
                (SELECT  L5.unit_task
                FROM     logic AS L5, logic AS L4, switch
                WHERE    SUBSTR ( L5.unit_switch, L4.comp_unit, 1) =
                         switch_id
                AND      L4.comp_unit IN
                    (SELECT  L3.comp_unit
                    FROM     logic AS L3, logic AS L2, switch
                    WHERE    SUBSTR( L3.comp_switch, L2.reqt_comp, 1) =
                             switch_id
                    AND      L2.reqt_comp IN
                        (SELECT  L1.reqt_comp
                        FROM     logic AS L1, switch
                        WHERE    SUBSTR( L1.reqt_switch, 1, 1) =
                                 switch_id)))))) @
```

The "**1**" in the line above supplied by the user out of the other values of "**2**" or "**3**". The output tests correctly as:

```
step_item
- - - - - - -
        2
        1
        1
        3
```

Generic SQL Code for Layered Logic Tables

The generic SQL code for layered logic tables 1-8 is below. The sequence of these correlated queries returns the procedures and respective procedure switches for updating.

```
SELECT …
FROM …
WHERE SUBSTR( switch, index_value, 1) = [ valid_switch ]
AND index_value IN
  (SELECT  index_value_passed
  FROM …
  WHERE SUBSTR( alias switch, user_input_index, 1) = [ valid_switch ] )
```

Conclusion

LLT is ideally suited for programming looms in the weaving industry and by extension for implementing software factories, manufacturing, dispatch scheduling, and networks. The generalized implementation is:

```
IN ( SELECT … FROM … WHERE SUBSTR … = [ valid_switch ])
```

Acknowledgements

Thanks are due for helpful discussions to LR Cagg, MD Mousa, and DL Vulis.

References

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.