# Real time Performance Arrives: Logic Table Technology for DB2

COLIN JAMES III

**How logic table technology coerces SQL to do procedural processing with real time performance for DB2**

In 1973 IBM invented Structured Query Language (SQL) to access their relational database systems which were based on a prefect mathematical model. Some report that SQL is now the most common computer language in use today, eclipsing newer C and older COBOL. But SQL is viewed as a *data access* language rather than a higher order *procedural* language which can loop and branch. That was the case until now.

Recently it was discovered how to coerce the non procedural SQL to perform procedural processing. The value of this innovation is realized in the implications which follow such as: perfectly portable source code; maintenance of one set of source code over diverse platforms; direct mapping and simple implementation of complex business rules; and real time relational database performance.

## PROCEDURAL PROCESSING WITH SQL

To automate relational database access, the method of choice was to embed ad hoc SQL queries within procedural languages. Some of these such as Ada, C, COBOL, FORTRAN, PL/I, and PL/SQL support iteration and decision paths. Using logic tables and SQL triggers makes it possible to coerce SQL to perform as a high order procedural language. This avoids reliance on embedded SQL statements which are universally difficult and not portable.

Logic table technology (LTT) is made up of special purpose relational tables and SQL triggers which work together. The triggers read the logic table which in turn instructs the relational database engine what actions to perform next. How many times such an action is performed is determined by the trigger's constraints or WHERE clauses.

The logic table contains two columns. One column is for row numbers. Another column is for fixed length strings of characters. Each character acts as a logic switch for its positional location within the string. It is from these logic switches that LTT derives its name.

The SQL triggers sequentially fire to read the logic table. The triggers make use of the substring or SUBSTR function in SQL to scan all of the character strings at a certain position of interest for the logic switch of interest. The query returns those row numbers having the particular logic switch of interest at the particular position of interest within the strings. Those row numbers then instruct the relational database engine how to proceed.

## A SAMPLE ACCOUNTING SYSTEM

The structure and content of LTT is illustrated in an accounting system named Report Accounts (RA). The logic table contains a column of row numbers representing numbered accounts. The logic table also contains a column of fixed length character strings or logic switches. Each switch represents by its positional number the number of a type of accounting transaction. Each switch also represents by its character content the particular type of accounting action to be taken. In double entry bookkeeping, the types of action are credit (Cr), debit (Dr), or none. In triple entry bookkeeping, another type of action is trebit (Tr). RA supports N-entry bookkeeping where any number of types of actions are available.

Here is a sample scenario in six steps of how LTT works with accounting arithmetic. See the logic table in Fig. 1. The user enters a dollar amount and transaction type into a screen input box. The SQL triggers then automatically fire to do the following.

1. The transaction type is looked up for its associated transaction type number.
2. The logic table string of characters is indexed to the position corresponding to that transaction type number.
3. The SUBSTR function searches that position in all character strings to find any logic switches set to not null.
4. The query returns all row numbers with that condition. As each row number is returned, the corresponding account number is updated with the dollar amount by the action specified in the logic switch.
5. A record of the account transaction is inserted in a transactions table.
6. The running balance of the account is updated in a balance table. This avoids searching the entire transactions table for the running balance of an account.

| Rows indexed as account numbers | Logic as transaction type *Position* | | | | | | | ↓ |
|---|---|---|---|---|---|---|---|---|
| | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
| 100 ← | | Cr | | Dr | | Dr | | Cr |
| 101 | Dr | | Cr | | Dr | | Cr | |
| 102 ← | | Cr | | Dr | | Cr | | Dr |

Transaction type 8 returns the logic to credit account 100 and debit account 102.

**Fig. 1: Accounting Logic Table**

For audit tracing, the transactions table contains all transactions made and grows infinitely. For corrections, a transaction reverses a previous transaction.

RA also supports all possible financial reporting instruments with the logic table. Report contents are specified as print fields in the logic switches. RA is very compact and implemented in 10 relational tables and 49 lines of SQL code.

The content of the logic table is user defined. RA is preloaded with logic for the accounting arithmetic of Standard General Ledger (SGL). That is required for accounting use by all departments of the US Government. SGL contains a base core of 152 complex transactions.

## 100% PORTABLE CODE

The SQL code used to access the logic table is contained entirely in triggers which are perfectly portable between relational database vendors. Moving code between vendors takes less than one hour due to the vendor specific syntax of the one line wrapper code around the triggers.

From the RA example, one trigger of 49 lines of code does all of this as follows: INSERT transactions (8 lines); UPDATE transactions (4 lines); INSERT accounts (13 lines); UPDATE balances (14 lines); and UPDATE transactions (10 lines).

Achieving the goal of 100% portable code means that one set of source code is maintained across implemented platforms.

## WHERE REAL TIME PERFORMANCE ROCKS AND RULES

Applications implemented using relational database technology are traditionally not real time performance critical. Examples are accounting and financial systems which process outside of business hours, much as banks credit deposits made after 3 PM onto the next business day.

However an increased demand for high performance is dictated by requirements for enormous gluts of incoming data. Examples are cellular telephone billing systems, subscriber program scheduling for High Definition Television (HDTV), and shop floor manufacturing where raw materials and resources are in contention.

To meet the demand for performance, LTT delivers real time performance from relational database engines. Performance is realized by compact database structure and access. The structure is extremely simple, such as the 10 tables in RA. Logic tables are small enough to be stored in the main computer memory at all times, thus speeding read response time. The access is very fast with small SQL triggers, such as the 49 lines of code in RA. Such native triggers also avoid pitfalls of SQL code embedded in high order languages.

## DB2 OUTPERFORMS THE FIELD

LTT was stress tested for performance on multiple relational database engines. The vendor products were DB2 UDB 5.2, ORACLE 7.2, and SQLServer 7.0. Each computer platform contained the identical NT operating system, processor speed, and memory capacity.

The sample accounting system above was used without reporting capability and thus with eight tables and seven triggers totaling 49 lines of SQL code. All tables were minimally populated with data and hence contained in the computer memory. The stress load was an average SGL transaction of 25 debits and six credits.

The results were astonishing. DB2 ran six times faster than ORACLE and 12 times faster than SQLServer.

## RECENT ADVANCES IN LTT

Structure of the logic table is critical to overall performance. Size and complexity of the logic table is of interest because a densely populated logic table occupies as much space as one sparsely filled.

The reader by now has inferred that complex business rules may be captured in multiple chained or connected logic tables. These logic tables may in turn index each other or themselves so as to self-modify. Multiple logic tables may be stacked then folded into each other to save space. To measure the complexity of chained logic tables formulas are developed to determine the total number and type of all possible query combinations.

The code structure of the SQL queries to access logic tables was implemented and generalized into Petri Nets (PN). It was discovered that the logic table queries match the common PN framework known as Kanban cells in flexible manufacturing systems (FMS). Therefore LTT is ideally suited as a basis for intelligent, automated software factories to supplant and replace the diverse yet flexible software functions for the various definitions of middleware.

## ACKNOWLEDGMENTS

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

**Colin James III is principal scientist at CEC Services, LLC. This article is based in part on his published conference papers and recent doctoral dissertation. His email address is cj3@cec-services.com.**