TransJoin: An Algorithm to Implement Division Operator of Relational Algebra in Structured Query Language

A Imamuddin[1], I Nahar[2], and S Chandra[2]

[1] Computer Science Department, Sekolah Tinggi Teknologi Muhammadiyah Cileungsi, Bogor, Indonesia
[2] Computer Science Department, BINUS University, Jakarta, Indonesia

*ashari@sttmcileungsi.ac.id

Abstract. Division operator is one of operators in Relational Algebra which is not implemented directly in SQL (Structured Query Language) standard. Therefore it is not be able to retrieve division query results with an SQL statement in the current relational database management systems (RDBMS). Database programmers have to create a complicated query to perform the task. It is the only relational algebra operators which is not implemented in ANSI SQL standard. This research aimed to study and design an algorithm named TransJoin (transformation and join) to implement it in SQL. TransJoin works to relation P(x, y) divided by Q(y) resulted R(x) through grouping and transforming each y attribute becomes P'(x, y') and Q'(y') and each x value is a single tuple with y' is a composite value of y. Then, result R(x) is resulted of joining P'(x, y') and Q'(y') relations by matching y' attribute. TransJoin was implemented in an open source RDBMS SQLLite. TransJoin tested and delivered valid results by comparing of traditional SQL queries and our proposed SQL queries with various data. The research showed that our algorithm is much more efficient by consuming 0.078 milliseconds compared contrasted the traditional query in 1,974.08 milliseconds for 9,991 tuples.
Key words: relational database; relational algebra; division operator SQL; algorithm; query processing;

1. Introduction

Modern database was started in 1960s when computer was discovered. In 1970, Edgar Frank Codd published a paper "A Relational Model of Data for Large Shared Data Banks". It was the first theory of relational database. Codd offered algebra as foundation of query language for database. He introduced six primitive algebra operators: Selection, Projection, Cartesian product, Union, Set difference and Rename. While the derivative operators are Intersections, Divisions, and Joins. Join operator consist Natural Join, Equijoin, Semijoin, Antijoin and Devide [1].

SQL (structured query language) is a relational database language. In 1986, a standard for SQL was defined by the American National Standards Institute (ANSI) and was subsequently adopted in 1987 as an international standard by the International Organization for Standardization (ISO, 1987). More than one hundred DBMSs now support SQL, running on various hardware platforms from PCs to mainframes [2].

SQL implemented all Code's relational algebra operations except division operations. Native SQL did not have direct operator to solve division query need. In the order words, no direct syntax of division operator in SQL. SQL programmers built their own methods and logics using existing operators to solve their requirements.

Cases of division operations are very common and often appear in queries used to solve any problems. However, the current solution or temporary solution used was to use a combination of various

existing operators with their respective different methods. Using the existing query programmers built their SQL query by using sub-query. The method implicated to poor performance and inefficient to produce query results. Moreover, the logic were really complicated.

2. Methods

The research focused on study of division operator in SQL and designed an algorithm of division operator to be implemented as SQL standard. The algorithm applied using C language in SQLite, one of open source RDBMS engine.

The algorithm would be validated by comparing query results of the new standard with the ones using conventional method. The research would design recommendation standard of using our division operator to SQL Organization (sql.org) to implement our research output.

3. Algorithm Design

Division operator was the only operator which did not support and be implemented in SQL Standard for ANSI SQL-2008. Joe Celko, a writer for some books about SQL, had been devoted himself more than 10 years in ANSI/ISO SQL Committee, and a contributor of SQL-89 and SQL-92 standard, in his electronic journal deplored division operator has not been adopted in SQL standard.[4]

Division is binary operation which is written as R ÷ S. The operation results are tuples in set R relates to set S as the followed instance:

R

ColA	ColB
F	1
F	2
F	3
E	1
E	3
S	1
S	2

S

ColB
1
2

$R \div S =$

ColA
F
S

Figure 1. Division operation example

In the real world there are many division cases. For instances, finding programmers who have skill set of all programming language, finding suppliers who provide all product items, and finding library members who borrow all computer books. Database practitioners use different method to solve division cases. Current solving method combines several algebra operators: projection, Cartesian product, set difference, and join. They also use sub-query to solve the problems which is lack and inefficient performance.

Division operator algorithm comprises definite division and partial division. Below is explanation of both algorithms.

3.1. Definite Division Algorithm

Given two relations R(X,Y) and S(Y), the results of R ÷ S operation defined as maximum relation Q which qualifies for:

$$Q(X) = \{ t \mid \text{for all } t_S \in S, \text{ there is a tuple } t_R \in R \text{ where } t_R.X = t \text{ and } t_R.Y = t_S \} \quad (1)$$

The definition is written as:

$$Q(X) = \{ t \mid (\{t\} \times S) \subseteq R \} \quad (2)$$

Furthermore, an algorithm which directly implements the above definition will be given, but beforehand it will be explained beforehand a definition that will be used in the algorithm:

Definition 1: For a relation R(X,Y), where X and Y may composite, R grouped by X, written as $\overset{X}{\Delta}(R)$, is a grouping operation R(X,Y) by X transforms to γ(X,Y) that:

$$\overset{X}{\Delta}(R) \equiv \gamma(X,Y) \equiv \{(\varkappa, \Upsilon(\varkappa)) \mid (\varkappa \in r.X) \wedge (\forall t \in R)\,((t.X = \varkappa) \Rightarrow t.Y \in \Upsilon(\varkappa))\} \quad (3)$$

The following is an example for R(X,Y):

R

X	Y
a	x
a	z
d	w
d	x
d	z

then, $\overset{X}{\underset{Y}{\Delta}}(R)$ is:

X	Y
a	{x,z}
d	{w,x,z}

Algorithm 1: Algorithm to derive results of R(X,Y) ÷ S(Y)
***Input*:** relation R(X,Y) and S(Y) without redundand tuple and X and Y may composite
***Output*:** output of R(X,Y) ÷ S(Y)

1. Result = Ø;
2. R'(X,Y) = R(X,Y) ⋉ S(Y);
3. $\gamma(X,Y) = \overset{X}{\Delta}(R'(X,Y))$;
4. For each tuple $t_i \in \gamma$ do {
5. If ($|t_i.Y| == |S|$) Result = Result ∪ $\{t_i.X\}$;
6. }
7. Return (Result);

3.2. Partial Division Algorithm
For two relations R(X,Y) and S(Z,Y), the result of R ÷ S is defined as maximum relation Q which qualifies for:

$$Q(X,Z) = \{\, t \mid (\{t\} \bowtie \overset{Z}{\Delta}(S)) \subseteq R \bowtie S \,\} \quad (4)$$

The following is given two algorithms to implement the definition of the relation structure of the results of the partial division operations using the help of **Definition 1** (transformation). In addition, one of them also uses **Algorithm 1** (Definite Division Algorithm):

Algorithm 2: Algorithm to derive results of R(X,Y) ÷ S(Z,Y)
Input: Relation R(X,Y) and relation S(Z,Y) without redundant tuple, X,Y and Z may composite
Output: results of R(X,Y) ÷ S(Z,Y)

1. Result = Ø;
2. $R'(X,Z,Y) = R(X,Y) \bowtie S(Z,Y)$;
3. $\gamma(X,Z,Y) = \overset{\{X,Z\}}{\Delta}(R'(X,Z,Y))$;
4. $\theta(Z,Y) = \overset{Z}{\Delta}(S(Z,Y))$;
5. For each tuple $t_i \in \gamma$ do {
6. For each tuple $u_i \in \theta$ do {
7. If ($t_i.Z == u_i.Z$ && $|t_i.Y| == |u_i.Y|$) Result = Result ∪ $\{t_i.X, t_i.Z\}$;
8. }
9. }
10. Return (Result);

In conclusion, the algorithm works by grouping (transforming) all values of each group into single tuple (cell) for both columns then performing inner join for value of both columns (attributes). We call this algorithm as TransJoin (transformation and join).

4. Proposal of New SQL Standard on SELECT Statement

In this section we will discuss grammar design and syntax of the algorithm. Division operator implemented with a syntactic pattern which is grammatically similar JOIN operator. As join operation, a division operation involves more than one table, and requires a column that is made as a condition (divisor factor attribute). We applied the design algorithm (Definite Division and Partial Division). The algorithm is implemented syntactically using the same the operator, DIVIDE.

Design of proposed SQL standard composed in the form of BNF (Backus–Naur form) to ease probability of existing RDBMS which support the form. Modification of SQL standard is by inserting in code of QUERY (SELECT STATEMENT) with addition of using DIVISION operation. The following is modification of the standard.

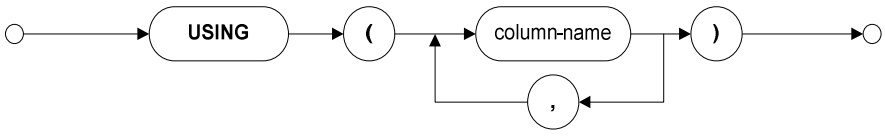
```
…
<scalar subquery> ::= <subquery>
<subquery> ::= <left paren> <query expression> <right paren>
<query expression> ::= <non-join query expression> | <joined table> |
<divided table>
<non-join query expression> ::= <non-join query term>
  | <query expression> UNION [ ALL ] [ <corresponding spec> ] <query term>
  | <query expression> EXCEPT [ ALL ] [ <corresponding spec> ] <query
term>
<non-join query term> ::= <non-join query primary>
  | <query term> INTERSECT [ ALL ] [ <corresponding spec> ] <query primary>
<non-join query primary> ::= <simple table> | <left paren> <non-join query
expression> <right paren>
<simple table> ::= <query specification> | <table value constructor> |
<explicit table>
<query specification> ::= SELECT [ <set quantifier> ] <select list> <table
expression>
<select list> ::= <asterisk> | <select sublist> [ { <comma> <select
sublist> }... ]
<select sublist> ::= <derived column> | <qualifier> <period> <asterisk>
<derived column> ::= <value expression> [ <as clause> ]
<as clause> ::= [ AS ] <column name>
<table expression> ::= <from clause> [ <where clause> ] [ <group by clause>
] [ <having clause> ]
<from clause> ::= FROM <table reference> [ { <comma> <table reference>
}... ]
<table reference> ::= <table name> [ <correlation specification> ]
```

```
  | <derived table> <correlation specification> | <joined table> | <divided
table>
<correlation specification> ::= [ AS ] <correlation name> [ <left paren>
<derived column list> <right paren> ]
<derived column list> ::= <column name list>
<derived table> ::= <table subquery>
<table subquery> ::= <subquery>
<joined table> ::= <cross join> | <qualified join> | <left paren> <joined
table> <right paren>
<cross join> ::= <table reference> CROSS JOIN <table reference>
<qualified join> ::= <table reference> [ NATURAL ] [ <join type> ] JOIN
<table reference> [ <join specification> ]
<join type> ::= INNER | <outer join type> [ OUTER ] | UNION
<outer join type> ::= LEFT | RIGHT | FULL
<join specification> ::= <join condition> | <named columns join>
<join condition> ::= ON <search condition>
<named columns join> ::= USING <left paren> <join column list> <right
paren>
<join column list> ::= <column name list>
<divided table> ::= <table reference> DIVIDE <table reference> <divide
specification>
<divide specification> ::= USING <left paren> <column name list> <right
paren>
<where clause> ::= WHERE <search condition>
<group by clause> ::= GROUP BY <grouping column reference list>
<grouping column reference list> ::= <grouping column reference> [ {
<comma> <grouping column reference> }... ]
<grouping column reference> ::= <column reference> [ <collate clause> ]
<collate clause> ::= COLLATE <collation name>
<collation name> ::= <qualified name>
<having clause> ::= HAVING <search condition>
<table value constructor> ::= VALUES <table value constructor list>
<table value constructor list> ::= <row value constructor> [ { <comma>
<row value constructor> }... ]
<explicit table> ::= TABLE <table name>
<query term> ::= <non-join query term> | <joined table> | <divided table>
<corresponding spec> ::= CORRESPONDING [ BY <left paren> <corresponding
column list> <right paren> ]
<corresponding column list> ::=  <column name list>
<query primary> ::= <non-join query primary> | <joined table> | <divided
table>
```

The following diagrams are BNF grammar to represent the flow language flow.

DIVIDE-OP

DIVIDE-CONSTRAINT



The following is given 3 (three) common relations in database system text books: Supplier, Part, and SupplierPart.

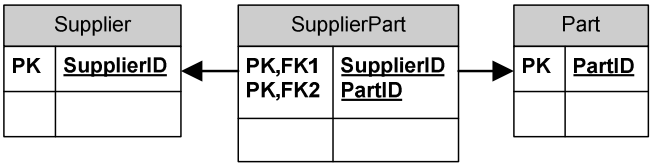


Figure 2. Supplier, part, and supplier part relations

Here is division operation of Relational Algebra to solve problem for finding suppliers (SupplierID) who supply all parts (PartID).

```
SuppliertPart ÷ Part
```

Using the existing standard we write below query statement.

```
SELECT DISTINCT SupplierID
FROM SupplierPart as sp1
WHERE NOT EXISTS
    (SELECT * FROM Part as p1
     WHERE NOT EXISTS
     (SELECT * FROM SupplierPart sp2
      WHERE sp1.SupplierID = sp2.SupplierID
           and sp2.PartID = p1.PartID))
```

While using the new standard the query statement is:

```
SELECT DISTINCT SupplierID
FROM SupplierPart DIVIDE Part using (PartID)
```

5. Implementation

The algorithm is implemented in SQLite (http://www.sqlite.org) which is an embedded RDBMS engine in the form of an in-process library that was developed using the C programming language. The main reason for choosing SQLite is because it is an RDBMS engine that is public domain and open source, so it is more realistic to be implemented as an object of implementation of the algorithm. The implementation is by modifying the SQLite library in order to make the engine recognize the "DIVIDE" token with the grammar-compliant syntax and produces output that matches the division operation.

Figure 3 depicts SQLite architecture. “Other source files” comprises C code files collection which contains 107 files including header-files. One of the files is “SELECT.C” as the core of SELECT statement. Based on an analysis of the overall flow of the existing process, the modification steps taken are: adding DIVIDE keywords to be recognized by the parser; modifying the grammar on the parser so that it can recognize DIVIDE syntax patterns; adding a new type other than JOIN and modify the parser to generate code for that type; and adding new process functions as the core of the processing routines performed on the DIVIDE type..

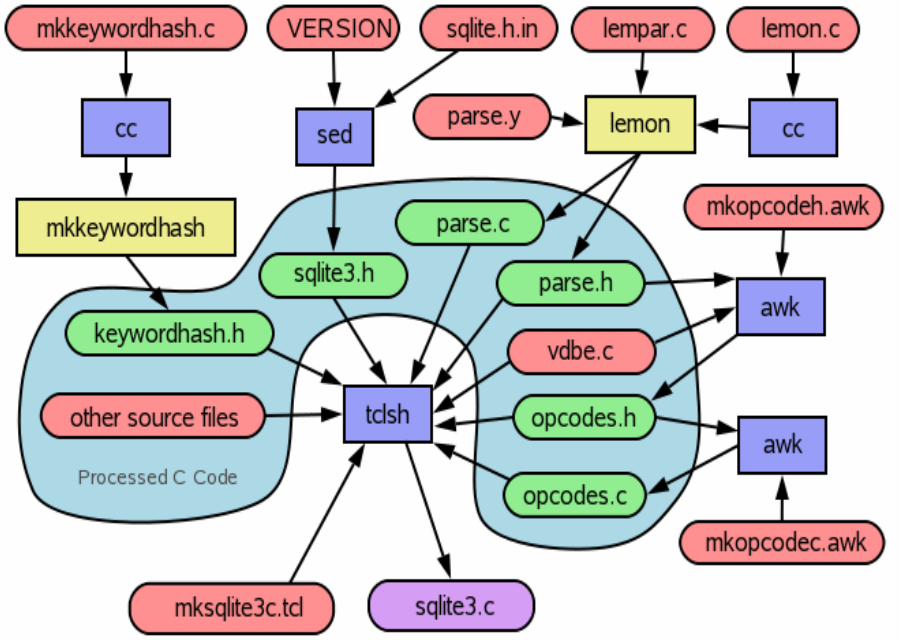


Figure 3. SQLite architecture

6. Evaluation

Tests for the proposed new query (SELECT query contains the DIVIDE operator) compared against classic queries which are alternative queries that exist in a variety of commonly found database literature books [5][6][7][10] and in general are often used and taught academically as an alternative form of query for the implementation of division operations.

The test factors to be compared are of two types. The first one is in terms of performance factors to the existing query by looking at the travel time used (elapsed time). Then the second is in terms of ease of use factor with the indicator used is the number of tokens / words used in each query. We conducted 10 (ten) tests with different number of tuples then we compare query time between the conventional query and new query for the same cases. Q1 is execution time for conventional query while Q2 is for the new query as in Table 1. The test used the same machine with the specifications are Intel Core2Duo 2,66Ghz, RAM 3GB under Windows 7 32-bit operating system.

Table 1 showed that that new query (using our algorithm) is much more efficient by consuming 0.078 milliseconds compared contrasted the traditional query in 1,974.08 milliseconds for 9,991 tuples.

Table 1. Test results

Test#	Supplier Count	Part Count	SupplierPart Count	Result Count	Q1 Time	Q2 Time
1	175	35	2061	39	35.708	0.016
2	200	40	2865	51	82.992	0.016
3	225	45	3241	47	102.774	0.016
4	250	50	4152	59	209.15	0.031
5	275	55	5064	66	345.869	0.032
6	300	60	5185	58	359.441	0.032
7	325	65	7398	86	927.207	0.032
8	350	70	7518	75	968.906	0.063
9	375	75	9163	91	1515.47	0.063
10	400	80	9991	81	1974.08	0.078

7. Conclusion

This research has produced division operator algorithm which can be used as an alternative in answering the problem of division operations in SQL. The designed algorithm can be implemented on the RDBMS engine. Based on the test results, the algorithm that has been implemented on one of the RDBMS engines, namely SQLite and it is much more efficient to solve division operation problems by making queries containing subqueries) which is 120 times faster. Or in the other word, the proposed query is 77% shorter than the conventional solution.

The research also proposed new SQL standard to solve division problem. The proposed standard and algorithm are enable to be implemented in other RDMBS platforms.

References

[1] Codd, E.F (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM Volume 13, No. 6.
[2] Connolly, Thomas.M, Begg. Caroline. E. (2015) Database Systems. Sixth Edition, University of Paisley, London.
[3] Baase, Sara. (1999). Computer Algorithms: Introduction to design and analysis. Third Edition. Addison-Wesley, California.
[4] Celko, Joe. (2005). SQL For Smarties: Advanced SQL Programming. Third Edition. Morgan Kauffman, California.
[5] Date, C.J (2004). An Introduction to Database System, 8th Edition. Addition Wesley Publishing, Massachusetts.
[6] Desai, B (1990). An Introduction to Database System, 7th Edition. West Publishing CO, Minnesota.
[7] Elmasri R., Navathe S.B. (2004). Fundamentals of Database System. Fourth Edition. Addison-Wesley, California.
[8] Fortier, Paul J. (1999). SQL-3 Implementing the Object Relation Database. McGraw-Hill, New York.
[9] Groff, James R. (1999). SQL: The Complete Reference. McGraw-Hill, New York.
[10] Kroenke, David. (2006). Database Processing Fundamentals, Design and Implementation. Tenth Edition. Pearson Prentice Hall, New Jersey.
[11] Mannino, Michael V. (2004). Database, Design, Application, Development & Administration. Second Edition. McGraw-Hill, New York.
[12] O'Neil, Patrick. (1999). Database Principles, Programming, Performance. Morgan Kauffman, California.
[13] Ramakrishnan R., and Gehrke J. (2000). Database Management Systems. Second Edition. McGraw-Hill, New York.
[14] Tseng, Frank S.C, Arbee L.P. Chen, Wei-Pang Yang. (2000). Implementing the Division Operation on a Database Containing Uncertain Data. Future Databases, Kyoto.
[15] Watson Richard (1999). Database Management – Databases and Organization. Second Edition. Wiley, USA.