Pinto, Paulo Jorge Gonçalves, 1956-
Yang, Hongji

Definition of sort function in relations and its usage in relational database management systems

http://hdl.handle.net/11067/5220

Metadata

Issue Date 2010

Abstract Este documento demonstra que o modelo matemático obtido pela adição de uma função a uma relação, e que a possa ordenar, demonstrando que a propriedade de fecho das operações entre relações se mantém, pode constituir uma extensão do modelo relacional....

This document approaches the possibility of adding a mathematical function to a relation that could sort that relation, demonstrating that the closure property of relations are still kept, so that this mathematical model can be used as extension of the base relational model....

Keywords Bases de dados relacionais

Type article

Peer Reviewed No

Collections [ULL-FCEE] LEE, n. 10 (2010)

DEFINITION OF SORT FUNCTION IN RELATIONS AND ITS USAGE IN RELATIONAL DATABASE MANAGEMENT SYSTEMS

Paulo Jorge Gonçalves Pinto
Doutorando em Base de Dados (DeMontfort University)
Docente da Universidade Lusíada

Hongji Yang
Ph.D. (Durham), MIEEE
Professor em DeMontFort University

Resumo: Este documento demonstra que o modelo matemático obtido pela adição de uma função a uma relação, e que a possa ordenar, demonstrando que a propriedade de fecho das operações entre relações se mantêm, pode constituir uma extensão do modelo relacional.

Abstract: This document approaches the possibility of adding a mathematical function to a relation that could sort that relation, demonstrating that the closure property of relations are still kept, so that this mathematical model can be used as extension of the base relational model.

1. Introduction

When E. F. Codd, an IBM researcher, in late 1968 defined the mathematical model known nowadays as the relational model, and in June 1970 published "A Relational Model of Data for Large Shared Databanks"[1] – document that is considered a classic in the present – what really happened was that, differently from former systems[1], Codd gave rigor and principles to his model before the implementation.

Since then several authors that, in a direct or indirect form were based on this document (and in others that followed) and the industry used such model as a base to implement RDBMS (Relational Database Management Systems). Are known, besides a second Codd document published in 1990, several books from other several authors like C. J. Date[4,9].

In the relational model, and according to Codd's paper, any user would always see the data through the shape of a table, even if that data were the result of a former relational operation with another set of data. This way has the property of being simple and effective, since the outcome of a relational operation can be used seamless as an input to another relational operation, with no need for the user to perceive it.

What wasn't approached on the model it's the utility that is given to each one of the relations and although we can say that for a relation the order of the rows it's not concerning (because it contains the same data), to an end user that isn't true because he will use it differently.

Even Codd, in his original paper nothing states about the row order (besides it's irrelevant), since that order seems not important to the development of relational algebra.

But let us look at some theoretical aspects. Relational Algebra, because its an algebra in a mathematical way, considers that for a certain set (the relation set) and for certain sets of operations (called relational operations) it has the closure property, meaning that the result of an operation over the relations would produce a new relation itself contained in the original set of relations[13].

[1] The systems that were in use at that time were the inverted list, hierarchic and network. The models for those systems were defined afterwards and not before they were commercially implemented. The relational model was first defined and then implemented.

Is this property that allows to present all the relations in a shape of a table, and allows also to define those operations, not as row to row processing process, but as a result set of an operation between relations (although the row to row processing would be laying under that, but its responsibility of the database engine, not the user's).

In an approach to the reality, we see that, beyond all, the order that the tuples from a relation present themselves to the end user, although they have no meaning to the relational calculus, they are indeed important to that user.

In a matter of fact, it's the utility that makes those data sets different, although they contain the very same data. It's the use that one user gives to those lists that makes them different from each other. A customer list sorted by customer's last name it's used differently from a list sorted by customer's ID, even if those two lists are, in a relational point of view, the same relation.

We are to believe, then, that the order of the rows it's not so irrelevant that can be depicted.

Let us take as an example, an application designed with database technologies that has as a fundamental request that the out coming data of a certain query should <u>never</u> be presented in the same sort order although the base query the same. This is a real case and concerns a list of hotels with rooms to let and the request is that even if two different users choose the same criteria to search for a room, the result must be sorted differently so that equality of opportunity was given to each hotel in the list. How can we fulfil that request if sort order is not a result of a relational operation?

In another case it is necessary to obtain the values of a budget with the running sum along with it (in every row, not summarized at the end). How can we sum in each row the precedent values without breaking the database performance, meaning, without having to calculate totals for the running sum for every row in the table? How can we do this if we have no order and we cannot know which row is before and after in the result set?

As these there are hundreds of other similar cases where sorting the results has a major importance in the final outcome.

With these considerations, we must conclude that the relational model will be richer if we could implement sort order anyway.

But what is sorting a relation all about?

Sorting a relation is no more than assign an *order number* (that will start from 1 to the number of tuples of that relation) to every tuple of a relation and sort by that number. To do such as assignment we will have to define one function (the *sorting function*) that for each tuple returns a number between 1 and N, being N the number of tuples in that relation.

We will define, then, that one extended relation is composed by a set of data (a relation - R) and one sorting function . Since the function only applies to one relation (with a relation becomes an element of this new set), every relation will have one or more functions that can sort it. But the some relation composed with two different sorting functions are also two different elements of the new set.

If we operate that relation with another relation through any relational operator we will have a new relation (closure property of the relations) that will present its data with any kind of order even if there is no previous specification of that order.

But that result set will have some kind of order, so we can claim that such sorting function exists (although it might be unknown) since the results are presented in some kind of order.

Then this extended model also has a closure property as long as we take the relations (regardless of their sorting functions) operate them with a relational operator and assign an sorting function to the resultant relation. From a mathematical point of view it is formally an algebra.

On the other hand the sorting function must have a set of attributes to be considered as one.

A function to be considered an sorting function must:

- Accept a relation as its primary input
- Return the same relation with an extra attribute set: Order Id, which is a natural number starting in one and N, being N the number of elements (tuples).

With these two premises taken, we can consider the following

Considering the pair E1=(R1, 1) and E2=(R2, 2) with (R1, 1) being an extended relation in which R1 is a relation associated with an sorting function 1 and (R2, 2) is another extended relation in which R2 is a relation associated to an sorting function 2, we can define the relational operation θ between E1 and E2 (E1 θ E2) giving the result as an extended relation E3 = (R3, 3) in which R3 = R1 θ R2 and 3 its sorting function.

We will have then (E1 θ E2) = (R1, 1) θ (R2, 2) = ((R1 θ R2), 3) = E3

We shall note that the sorting function 3 doesn't have to be defined by the other two (1 and 2).

Then how could this help to solve the problems that were stated at the beginning of this paper?

Let's see. In the first case, it would be sufficient to associate to the resulting relation a sorting function with a random output, so that each time the function was associated with that particular set of data would produce a different extended relation.

In the second case, since the sorting function can be inverted it's not only possible to know what particular order has a tuple, but also its possible to know the value of any field from any row. We could have access to the running value stored in the previous row and, without remaking all the calculations, to add only the value in a line to the previous running values to obtain the new running value.

We must now perceive what the advantages of this model are since the industry, regardless of the theoretical model, has implemented its own data sorting (need felt since ever) through SQL[7], being SQL itself a standard (ORDER

BY clause).

What then this new model brings that SQL's ORDER BY doesn't have?

First of all, the ORDER BY clause can only be applied to the columns (fields) of the result set, whistle the sorting function, although it receives as an argument the all tuple, doesn't need to use any of the raw data to assign a order number (e.g. The "natural" order, meaning, whatever tuple is presented the sorted results would always be 1,2,3,…,N).

On the other hand, the ORDER BY clause uses only the binary content of the field to sort the results out (ascending or descending), whistle the sorting function doesn't have that limitation. It can assign an order number to a row depending only the way it was defined.

Finally we can use the sorting function to define non-linear business rules, complex mathematic expressions, random expressions or any other kind of expressions, since the model have no restriction in its internal logic and how the sorting numbers are calculated.

So, in the first case we could only execute a simple database query with the following *possible* syntax:

```
SELECT HotelName, RoomFee
FROM Hotel
WHERE RoomType IN (@TypeList) AND RoomFee BETWEEN @LowPrice
AND @HighPrice
ORDER BY RANDOM()[2]
```

Since that statement its not possible to issue, with the present database engines, it was build a procedure that assigns to a temporary table to store the results a new field filled with a random number between 1 and ROWCOUNT() (i.e. the number of rows the the query is returning). Finally the procedure queries that temporary table sorting out the results by that random field.

This is a so complex solution that it required a row level processing which must be avoided at all cost.

Since row level processing must be avoided, row identification is nevertheless useful and can be used to avoid some of that nasty row level processing.

We now have, indeed, the concept of "first", "last", "previous", "next", "n-th" when referring to any tuple.

Although it seems like a going back, in concern with the relational model – in a matter of fact, after have been defined the operations between relations with the due independence regarding its tuples – getting row references (pointers) it seems like a going back. And it would be if we use it for row level processing of the relation. But what we want to define is some new aggregate functions that can use the relative positioning of the rows (their order) in a way that they could be integrated in the SQL so data can be manipulated.

One practical example would be the function PREV(domain) that

[2] We shall use the bold formatting whenever proposed syntax is included in SQL statements

would return the value in the domain “domain” in the previous row[3].

If we define that PREV(domain), according with other SQL aggregate functions, would return a NULL value for the first row, we could have the following query (in MS-SQL) to solve the second case:

```
SELECT DtBudg, MonthValue, ISNULL(PREV(Accumul),0) + MonthValue as
Accumul
FROM Budget
WHERE DtBudg BETWEEN '1-Jan-2004' AND '31-Dec-2004'
ORDER BY DtBudg()
```

DtBudg() is a sorting function based on a table field (it would be equivalent to the actual ORDER BY DtBudg).

This query would then give for a given time slice, the monthly values of the budget, sorted by budget date with the running sums of the monthly values along with these.

Given the following table

DtBudg	MonthValue
31-10-2003	1.000,00€
1-1-2004	500,00€
1-6-2004	2000,00€
1-1-2005	750,00€

The result of such a query would be:

DtBudg	MonthValue	Accumul
1-1-2004	500,00€	500,00€
1-6-2004	2000,00€	2.500,00€

The SQL to execute the very same function, using Microsoft’s T-SQL, which also solves the proposed case, is the following:

```
SELECT B.DtBudg, B.MonthValue, Accumul = (SELECT SUM(MonthValue)
        FROM Budget BB
        WHERE BB.DtBudg BETWEEN '1-Jan-2004' AND B.DtBudg)
FROM Budget B
WHERE B.DtBudg BETWEEN '1-Jan-2004' AND '31-Dec-2004'
ORDER BY B.DtBudg
```

Although, technically it can be executed, in a matter of fact, beyond the statement is much more difficult to understand, the sum is calculated for each row instead of using the last value computed from the last row, as our model proposes, and affecting the overall performance.

[3] Note that it is not the previous value for that domain, but the value for that domain from the previous tuple.

2. Conclusions

Data sorting has always been a need by those who use Relational Database Management Systems, that even in the standard SQL/92 was included the ORDER BY clause which allows to sort the resulting dataset of an SQL expression, but lacks theoretical support since the classical relational model disregard relation sorting because it's considered unnecessary.

Using such an extension could provide some theoretical support in a way to suppress the gap between the relational model most rigorous mathematical definitions and the standard SQL in what sorting is concerned.

This document only wants to be the very beginning of a deep study in which can be possible to define at the mathematical level a solid foundation to data sorting that, and similarly to the relational model itself, can transpose to the SQL a whole new set of functionalities related with sort orders and supported by this extension of the relational model.

Finally its clearly demonstrated that there is a need for, in this particular field, a more in-depth research is such a way that the gap referred here between the theoretical model and its practical implementation could be fulfilled

References:

[1] CODD, Edgar F.: "A Relational Model of Data for Large Shared Data Banks", Comm. of the ACM 13, No. 6 (June 1970)

[2] CHEN, Peter P-S.: "The Entity-Relationship Model - Toward a Unified View of Data", ACM, Transactions on Database Systems (1976)

[3] MARTIN, James: "Principles of Database Management", Prentice-Hall (1976, 1989).

[4] DATE, Christopher J.: "An Introduction to Database Systems - 8th Edition", Addison-Wesley (2003)

[5] NG, Wilfred K.: "An Extension of the Relational Database Model to Incorporate Ordered Domains", ACM, Transactions on Database Systems, Vol. 26, No. 3 (September 2001).

[6] REED, Paul: "The Unified Modeling Language Takes Shape", DBMS 11, Nº 8 (July 1998)

[7] ISO/IEC 9075-*: 2003, Information technology — Database Languages — SQL (2003~2006)

[8] – CODD, Edgar F.: "Domains, Keys, and Referential Integrity on Relational Databases", InfoDB3, Nº 1 (Spring 1988)

[9] – DATE, Christopher J.: "Referential Integrity", Proc.7th Int. Conference on Very Large Data Banks, Cannes, France (September 1981)

[10] – HALL, P. OWLETT, J. and TODD, S. J. P.: "Relations and Entities" in G. M.

Nijssen (ed.) Modeling in Data Base Management Systems, Amsterdam, The Netherlands: North-Holland/New York, N. Y.: Elsevier Science (1975)

[11] – CODD, Edgar F.: "Data Models in Database Management", Proc. Workshop on Data Abstraction, Databases and Conceptual Modeling, Pingree Park, Colo (June 1980)

[12] – CHAUDHURI, Surajit and SHIM, Kyuseok: "Optimizations of Queries with User-defined Predicates", Proceedings 22nd International Conference on Very Large Data Bases, Mumbai (Bombay), India (September 1996)

[13] – BRAUMANN, Pedro B.: "Teoria da Medida e da Probabilidade – Parte I: Álgebra de Conjuntos", Fundação Calouste Gulbenkian, (1987)[4]

[4] Professor Pedro Braumann was Portuguese and this book was never translated. The title means "Theory of measure and probability – Part I: Algebra of Sets"