14

The From Clause

Richard T. Snodgrass, Christian S. Jensen, and Fabio Grandi

1 Introduction

Information retrieval is an integral component of any database management system. Temporal database management systems should offer user-friendly and powerful means of retrieval of data according to temporal criteria. The From clause is an important component of the Select statement: it identifies the underlying relations from which the information is to be retrieved and allows the declaration of range variables. While variables merely serve as "correlation names" (e.g. for joining a table with itself) in SQL-92, TSQL2 variables are designed to increase the temporal expressiveness of the language, in addition to provide "syntactic sugar" in making some queries easier to formulate.

TSQL2 range variables generalize the concept of *history variables* [2], which reference "groups" of tuples with a common value of a time-invariant key or surrogate, as earlier proposed for HoTQuel [1]. The main extension concerns the possibility of grouping tuples on arbitrary sets of columns. This feature also generalizes the concept of a *restructuring operator* [3], which changes the key of a temporal relation, as proposed in TempSQL [4].

2 Informal Definition

Let us examine a few examples, to provide a very informal description. As will be seen, this is an extension of the previous syntax. The `Employee` relation, with `Name`, `Dept`, and `Salary` attributes, will be referenced in the examples. The clause

```
FROM Employee
```

is equivalent to `FROM Employee AS Employee`, which is equivalent to `FROM Employee(*) AS Employee`, which in turn declares a tuple variable named

Employee ranging over the relation Employee grouped on all of its attributes, specifying that in each tuple, each attribute will have exactly one value. This example illustrates how the new syntax is upward-compatible with the existing syntax, and also how snapshot reducibility could be proven. The clause

```
FROM Employee(Name) AS Emp
```

groups on the Name attribute. There may be many values for the Salary and Dept attributes within a single “grouped tuple”, but there will only be one value for the Name attribute. In fact, the Salary and Dept attributes are inaccessible through Emp. We’ll see shortly how to access such attributes.

When the tuple variable’s lifespan is referenced, say in a where clause, the lifespan is the union of the chronons of the BCDM tuples having the same value for Name that were collected together to form the grouped tuple. Only the attributes mentioned in the <coalescing attributes> can be referenced in the rest of the query.

Who has been on the payroll for more than five years?

```
SELECT Name
FROM Employee(Name) AS Emp
WHERE CAST(Emp AS INTERVAL YEAR) > INTERVAL ’5’ YEAR
```

Since the from clause is grouped on Name, the lifespan of the Employee tuple variable is the lifespan of that employee, and is a temporal element.

Who has worked in Toys longer than Di has made $20,000?

```
SELECT E.Name
FROM Employee(Name, Dept) AS E,
   Employee(Name, Salary) AS D
WHERE E.Dept = "Toys" AND D.Name = "Di"
   AND D.Salary = 20000
   AND CAST(E AS INTERVAL DAY) >
       CAST(D AS INTERVAL DAY)
```

Note that the lifespan of D (a temporal element) is all the times that there is a tuple with D.Name = "Di" and D.Salary = $20,000. This cannot be done easily in a period tuple-timestamped language that employs a weaker From clause.

Tuple variables can be associated with other tuple variables. The clause

```
FROM Employee(Name) AS E, E(Name,Salary) AS F
```

specifies that F is a tuple variable with two attributes, effectively synchronized with E on the Name attribute. As syntactic sugar, it is not necessary to mention the shared attributes, and hence this From clause is equivalent to

```
FROM Employee(Name) AS E, E(Salary) AS F
```

This clause defines a tuple variable E, grouped on Name, and a “coupled” tuple variable F, grouped on Name and Salary (since F is coupled to E, it inherits E’s grouped attributes). E will range over Employee, grouped on Name. Then, F will range over all the tuples of E that are grouped on both Name and Salary. The Name attribute will be the same for both E and F at any time, but the salary can vary.

E and F are linked in another way. If, for a particular E, there is no F that satisfies the where clause, then E is considered not to have satisfied the where clause. This will fall out of the semantics, which treats a <correlation name> that appears as a <table source> simply as additional equality predicates on the shared attributes. Hence, the above from clause is equivalent to

```
FROM Employee(Name) AS E, Employee(Name, Salary) AS F
WHERE E.Name = F.Name AND E OVERLAPS F
```

We now discuss the second parenthesized component, the <partitioning unit>. The clause

```
FROM Employee
```

is equivalent to FROM Employee AS Employee, which is equivalent to FROM Employee(*) AS Employee. Note that no partitioning is the default. The clause

```
FROM Employee(PERIOD) AS Emp
```

is equivalent to FROM Employee(*)(PERIOD) Employee AS Emp. This from clause first groups on all attributes of Employee, then partitions the resulting temporal elements into maximal periods, yielding tuple timestamping with periods. This generates many value-equivalent tuples, each associated with exactly one (maximal) period, for the purposes of the rest of the query. Note that this operation is free if an period-tuple-timestamped representational data model is used (but is nonetheless important semantically).

Consider query Q 2.1.3 from the test suite, “Who worked continuously in the Toy department for as long as Di?”

```
SELECT E.Name
FROM Employee(Name,Dept)(PERIOD) AS E,
   Employee(Name,Dept)(PERIOD) AS D
WHERE E.Dept = "Toys" AND D.Dept = "Toys"
   AND D.Name = "Di"
   AND CAST(E AS INTERVAL DAY) >=
       CAST(D AS INTERVAL DAY)
```

Many queries are interested in maximal periods, and so being able to partition a temporal element into such periods is highly useful.

3 Expressive Power

It turns out that coalescing attributes are syntactic sugar in TSQL2's data model. Specifically,

```
FROM Employee(Name) AS E
```

is equivalent to

```
FROM (SELECT Name FROM Employee) AS E
```

This is true whether Employee is a snapshot relation or a valid time relation. In the latter case, the projection does an automatic coalescing of temporal element timestamps.

References

[1] Grandi, F. and M. Scalas. "HoTQuel: A History-Oriented Temporal Query Language," in *Proceedings of the 5th IEEE Compeuro*. Bologna, Italy: May 1991.

[2] Grandi, F., M. Scalas and P. Tiberio. "A History-oriented Data View and Operation Semantics for Temporal Relational Databases," in *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*. Ed. R. T. Snodgrass. Arlington, TX: June 1993.

[3] Gadia, S. K. "Weak Temporal Relations," in *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM SIGAct-SIGMod. Los Angeles, CA: 1986.

[4] Gadia, S. K. and G. Bhargava. "SQL-like Seamless Query of Temporal Data," in *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*. Ed. R. T. Snodgrass. Arlington, TX: June 1993.