The From Clause in TSQL2

September 23, 1994

A TSQL2 Commentary

The TSQL2 Language Design Committee

Title	The From Clause in TSQL2
Primary Author(s)	Richard Snodgrass and Christian S. Jensen
Publication History	September 1994. TSQL2 Commentary.

TSQL2 Language Design Committee

Richard T. Snodgrass, Chair	University of Arizona
`rts@cs.arizona.edu`	Tucson, AZ
Ilsoo Ahn	AT&T Bell Laboratories
`ahn@cbnmva.att.com`	Columbus, OH
Gad Ariav	Tel Aviv University
`ariavg@ccmail.gsm.uci.edu`	Tel Aviv, Israel
Don Batory	University of Texas
`dsb@cs.utexas.edu`	Austin, TX
James Clifford	New York University
`jcliffor@is-4.stern.nyu.edu`	New York, NY
Curtis E. Dyreson	University of Arizona
`curtis@cs.arizona.edu`	Tucson, AZ
Ramez Elmasri	University of Texas
`elmasri@cse.uta.edu`	Arlington, TX
Fabio Grandi	Universitá di Bologna
`fabio@deis64.cineca.it`	Bologna, Italy
Christian S. Jensen	Aalborg University
`csj@iesd.auc.dk`	Aalborg, Denmark
Wolfgang Käfer	Daimler Benz
`kaefer%fuzi.uucp@germany.eu.net`	Ulm, Germany
Nick Kline	University of Arizona
`kline@cs.arizona.edu`	Tucson, AZ
Krishna Kulkarni	Tandem Computers
`kulkarni_krishna@tandem.com`	Cupertino, CA
T. Y. Cliff Leung	Data Base Technology Institute, IBM
`cleung@vnet.ibm.com`	San Jose, CA
Nikos Lorentzos	Agricultural University of Athens
`eliop@isosun.ariadne-t.gr`	Athens, Greece
John F. Roddick	University of South Australia
`roddick@unisa.edu.au`	The Levels, South Australia
Arie Segev	University of California
`segev@csr.lbl.gov`	Berkeley, CA
Michael D. Soo	University of Arizona
`soo@cs.arizona.edu`	Tucson, AZ
Suryanarayana M. Sripada	European Computer-Industry Research Centre
`sripada@ecrc.de`	Munich, Germany

Abstract

This document proposes syntax and informal semantics for an extended From clause in the Select statement.

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, which identifies the underlying relations from which the information is to be retrieved, is an important component of the Select statement.

2 Informal Definition

In the language extension to be discussed shortly, we adopted the following goals.

1. Extensions should be upward compatible with current SQL-92.
2. Extensions should be as minimal as possible.
3. As few reserved words as possible should be introduced.
4. Punctuation should be consistent with the rest of the language.
5. Extensions should be consistent and compatible with user-defined time syntax.
6. Cleanliness of the BCDM should be retained.
7. Defaults should be carefully chosen to reflect common usage and to enable a suitable reduction proof (see (1)).

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 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`, which is actually equivalent to `FROM Employee(*)(ELEMENT) AS Employee`. Note that `ELEMENT` 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 the 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.

4 Acknowledgements

Support was provided in part by the National Science Foundation under grant IRI-9302244, and in part by the AT&T Foundation. In addition, support was provided in part by the Danish Natural Science Research Council under grants 11–1089–1 SE and 11–0061–1 SE.

A Modified Language Syntax

The organization of this section follows that of the SQL-92 document. The syntax is listed under corresponding section numbers in the SQL-92 document. All new or modified syntax rules are marked with a bullet ("•") on the left side of the production.

Where appropriate, we provide disambiguating rules to describe additional syntactic and semantic restrictions. We assume that the reader is familiar with the SQL-92 standard, and that a copy of the proposal is available for reference.

A.1 Section 5.2 <token>

One reserved word was added.

<reserved word> ::=
• `ELEMENT`

A.2 Section 6.3 <table reference>

The production for the non-terminal <table reference> is replaced with the following. The first component can be more complex than a single <table name>, and multiple space-separated <correlation name>s are permitted.

<table reference> ::=
• <table source> [[`AS`] <correlation> { <correlation> }...]
• | <derived table> [`AS`] <correlation> { <correlation> }...
| <joined table>

The following productions are added. The first allows table references to be defined in terms of other table references. The rest serve to define <correlation modifier>.

<table source> ::=
- <table name> <correlation modifier>
- | <correlation name> <correlation modifier>

<correlation> ::=
- <correlation name> [<left paren> <derived column list> <right paren>]

<correlation modifier> ::=
- [<left paren> <coalescing columns> <right paren>]
  [<left paren> <partitioning unit> <right paren>]

<coalescing columns> ::=
- <column name> [{<comma> <column name> }...]
- | <asterisk>

<partitioning unit> ::=
- `ELEMENT`
- | `PERIOD`

Additional syntax rules:

1. <coalescing columns> of <asterisk> imply all the attributes of the <table name> or <correlation name>.
2. If the <coalescing attributes> are not present, then <asterisk> is assumed.
3. If a <correlation modifier> is applied to a <table source>, then a <correlation> is required.
4. If the <correlation modifier> is applied to a <correlation name>, then the attributes are drawn from the table upon which the <correlation name> is based, and augment those attributes associated with the <correlation name>. The latter attributes can be mentioned in this <correlation modifier>, but is not required.
5. If <partitioning unit> is not specified, then `Element` is assumed.

Additional general rules:

1. Let CM be the <correlation modifier>. Let CN be a <column name> contained in CM, and C be the column.

   Case:

   - If CM is associated with a <table name>, then let T be that table name. The table identified by T is the *ultimate table* of CN.
   - If CN is associated with a <correlation name>, then let D be that <correlation name>. The ultmate table of CN is the ultimate table of D.

2. C must be a column of its ultimate table.
3. Only those <column name>s indicated as <coalescing columns> are accessible via the <correlation name>.