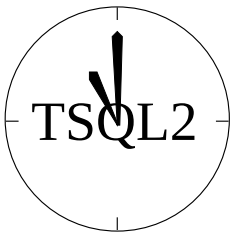

A Survey of Valid-Time Selection and Projection in Temporal Query Languages

September 16, 1994

A TSQL2 Commentary

The TSQL2 Language Design Committee

Title	A Survey of Valid-Time Selection and Projection in Temporal Query Languages
Primary Author(s)	Suchen Hsu, Christian S. Jensen and Richard Snodgrass
Publication History	May 1992. TempIS Document No. 30. 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

Copyright © 1994 Suchen Hsu, Christian S. Jensen and Richard Snodgrass. All rights reserved.

Contents

List of Figures

List of Tables

Abstract

Temporal databases have now been studied for more than a decade. During that period of time, numerous query languages have been proposed for temporal databases. One of the essential parts of a temporal query language is valid-time selection, which allows the user to retrieve tuples according to their valid-time relationship. Valid-time projection is another important ingredient which defines the timestamps of the tuples in query results. Here, nine different temporal query languages are examined with a focus on valid-time selection and projection. This document is intended to provide an ideal foundation for designing the valid-time selection and projection components of the consensus query language TSQL2 that is currently being designed.

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. Selection of tuples according to their *valid times*, which is the times when the information represented by the tuples is valid in the modeled reality [Snodgrass & Ahn 1985], is termed *valid-time selection.*

The attributes to be computed and returned from a query are specified in the target list. *Valid-time projection* is the temporal analog of regular projection in, e.g., QUEL, and allows for the specification of the implicit timestamps for tuples of the resulting relation.

Valid-time projection mechanisms may be partitioned into two categories. First, valid-time projection may be built implicitly into the operators and constructs of the query language. This leaves the user no freedom in specifying the timestamps of result tuples. Second, valid-time projection may be specified explicitly, as part of an existing clause or in a new clause. This approach allows the user to control the timestamps of result tuples.

In the past decade, numerous temporal extensions to the relational data model have been proposed. Most of these data model proposals have included user-level query languages. Most of these languages are not entirely new, but are instead extensions of existing, non-temporal query languages. To illustrate, extensions of SQL [Chamberlain et al. 1976] include TOSQL [Ariav 1986], TSQL [Navathe & Ahmed 1987, Navathe & Ahmed 1989], TempSQL [Gadia 1992], and HSQL [Sarda 1990A]; and extensions to QUEL [Held et al. 1975] include TQUEL [Snodgrass 1987], HQUEL [Tansel & Arkun 1986], and HTQUEL [Gadia & Vaishnav 1985, Gadia 1988]. The reason for this is the wide spread acceptance and usage of SQL and QUEL.

In the late 1970's and early 1980's, two important temporal data models were proposed, namely Ben-Zvi's TRM and Ariav's TOSQL. The designers of these languages did not place much emphasis on valid-time selection and projection; rather, they were interested in specifying the data model. As a result, the support for valid-time selection and projection is very limited in these languages. In the late 1980's and early 1990's, TSQL and HSQL were proposed. These languages have a clearer syntax than did their predecessors, and the the support for valid-time selection and projection is improved. Parallel to these efforts, temporal query languages based on QUEL were also being developed. These include TQUEL, HQUEL, and HTQUEL. Where TQUEL is based on 1NF temporal relations, the other two languages are based on non-1NF relations. Therefore, the syntax of HQUEL and HTQUEL is very different from that of conventional QUEL.

Our objective is to analyze the mechanisms for valid-time selection and projection in existing

temporal query languages. Such an analysis is a suitable foundation for designing valid-time selection and projection components of the TSQL2 query languge that are powerful and syntactically clear and thus make it easy to formulate and understand queries.

Before we proceed by surveying valid-time selection and projection in nine existing temporal query languages, we first define terminology that is useful when exploring the commonalities and differences between the languages. The language surveyed first is Legol 2.0, a pioneer temporal query language. Next, the four extensions to SQL are reviewed in chronological order. Then the three extensions to QUEL are examined, also chronologically. A final section contains a summary.

In examples throughout the paper, we use an employee relation, EMPLOYEE = (NAME, DEPT, POSITION), and the following natural-language query (in addition to other queries) is formulated in each of the covered languages.

Q1. List all of the employees who worked during all of 1990.

Also, the part of each language that relates to valid-time selection and projection is described precisely by means of a BNF syntax. As most of the languages do not define the temporal constants clearly, we use the same format, MM.DD.YY, for all languages. Coverage of temporal constants may be found elsewhere [Soo & Snodgrass 1992].

2 Survey

2.1 Terminology

Below, we define terms that are frequently used in the descussions of the existing language proposals. For clarity, we will use these definitions for all languages, even in situations where other terms were originally used.

Timestamp referencing This term denotes the method provided by the query language for refering to the timestamp values of tuples in queries (e.g., in temporal predicates and target lists).

Event extraction An interval consists of a starting event and an ending event. In temporal queries, it is often convenient to reference (e.g., for comparison) the events of intervals. Event extraction denotes the extraction of the delimiters of intervals.

Interval constructor As the opposite to event extraction, an interval constructor is an operator that creates an interval from events or intervals. One useful constructor takes two events as operands and constructs the interval with these two events as delimiters. Other constructors create new intervals from old intervals by intersection, extension, and union.

Event and interval comparison predicates Temporal query languages provide varying sets of built-in predicates for the comparison of temporal values. We distinguish between predicates for the comparison

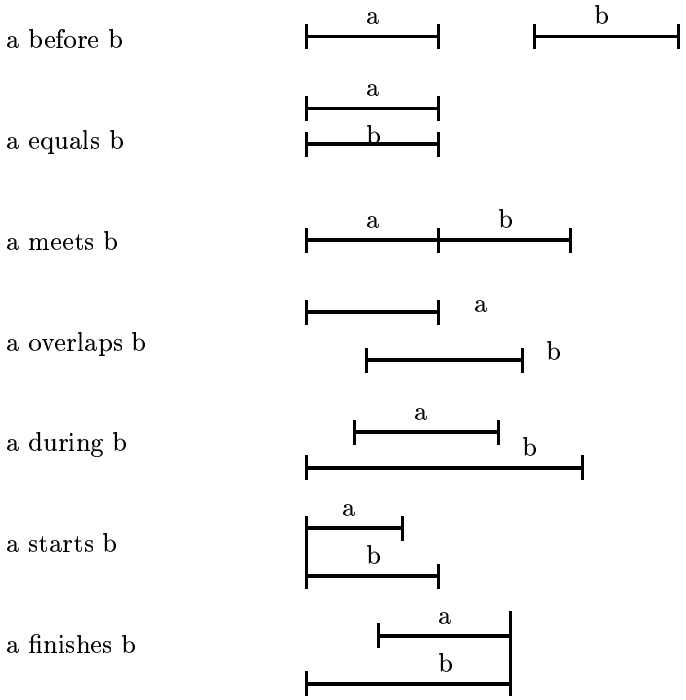

Figure 1: Allen's Interval Comparison Operators

of events and intervals. While some languages provide separate predicates for the two types of objects, other languages utilize the same predicates.

Temporal predicates By this we mean any boolean expression with comparisons involving time values. Temporal predicates are used for selecting tuples based on their timestamps.

As both events and intervals are present in temporal relations, temporal predicates may involve both types of timestamps. Events are totally ordered and may be treated similarly to integers and reals. Thus, conventional arithmetic comparison operators and logical operators are sufficient for event comparison. Regarding intervals, the comparsion operators of Allen have proved to be complete [Allen 1983]. Note that each of these operators can be simulated with event comparison operators, logical operators, and event extraction mechanisms. (Figure 1 shows the complete set of interval relations defined by Allen.)

Thus, languages that support event time comparison and event extraction have the same expressive power as languages that support interval comparison. While interval comparison operators thus do not increase the expressive power of a query language, such operators do improve the readability of queries.

Valid timeslice This operator retrieves the tuples from the argument relation which are valid during a specified interval or on a time point. Conceptually, it is like "cutting" a slice from a relation.

Temporal ordering A group of timestamped tuples may be numbered according to their timestamp order. By means of temporal ordering, tuples with particular numbers may be selected.

```
<source>              ::= <expression> {<operator> <expression>}
<expression>          ::= <simple expression> | "[" <source> "]"
<simple expression>   ::= <reference> | <literal> | <function call>
<reference>           ::= <entity reference> | <attribute reference>
<entity reference>    ::= <entity label> "(" <identifier list> ")"
<entity label>        ::= <alphabetic string>
<attribute reference> ::= <attribute label> of <entity reference>
<attribute label>     ::= <identifier label> | start | end
<identifier list>     ::= <identifier element> {"," <identifier element> }*
<identifier element>  ::= <identifier label> | <literal> | -
<identifier label>    ::= <alphabetic string> | <non-negative integer>
<literal>             ::= <quoted string> | <numeric constant>
<function call>       ::= <function name> [ for <control list>] of <expression>
<control list>        ::= <attribute label> {"," <attribute label> }*
<function name>       ::= <select> | <aggregate> | <value>
<select>              ::= max | min | highest | lowest | first | last | current | past
<aggregate>           ::= sum | number | accumulate | count | whenever
<value>               ::= duration | today
<operator>            ::= <setop> | <timeop> | < arithmeticop> | <comparison>
<setop>               ::= union | is | isnot
<timeop>              ::= while | while not | or while | during | since | until
<arithmeticop>        ::= + | - | * | /
<comparisonop>        ::= = | ≠ | > | < | >= | <=
```

Figure 2: BNF Syntax of Legol 2.0

2.2 Legol 2.0

Legol 2.0 was the first relational query language to support temporal queries [Jones et al. 1979]. This language supports tuple-timestamped valid-time relations. Specifically, tuples are assigned two implicit valid-time attributes, `start` and `stop`. These attributes are accessed by the various set-theoretic, temporal, and comparison operators. The query language is closed: The results of queries are valid-time relations. As a result, queries may be nested arbitrarily.

Legol 2.0 differs from other languages in that it is a rule-based, procedural query language. As our interest is declarative query languages, we examine Legol 2.0 only briefly.

In addition to the conventional set-based operations, Legol 2.0 introduces several temporal operations, including valid-time intersection, one-sided valid-time intersection, valid-time union, valid-time difference, and valid-time set membership. The semantics of these operators were explained by examples.

Next, temporal projection is implicitly embedded in the operators. For example, the valid-time intersection operation, implements interesction of timestamps, i.e., the valid time of an output tuple is the intersection of the valid times of the two input tuples. The semantics of these set operations is not clearly implied by the syntax (shown in Figure 2). As a result, it is hard for users to understand queries by simply reading them.

Tuple-variable names are used for implicitly referencing the timestamps of the tuples, and starting and ending events of intervals may be retrieved by the functions, `start of` and `end of`. The arithmetic comparison operators (>, >=, <, <=) are applicable also to events. While both event extraction and comparison are supported, Legol 2.0 lacks interval comparison operators and boolean operators.

Temporal ordering functions, such as `first, last, current and past`, are provided.

The sample query Q1 is expressed as follows.

```
EMP1990() ⇐ [ start of EMPLOYEE(NAME) < 12.31.90] while
            [ end of EMPLOYEE(NAME) > 1.1.1990]
```

In Legol 2.0, valid-time projection is built-into the operators. Some operators compute new timestamps by intersecting existing timestamps. Other operators define valid-time projection differently, the particular choice depending on the semantics of the operators.

2.3 Ben-Zvi's TRM

Ben-Zvi was the first person to propose a temporal data model with three different times, termed *effective time, registration time*, and *deletion time* [Ben-Zvi 1982]. Tuples have five implicit timestamps. The effective time is similar to valid time, and like Legol 2.0, tuples have two of these. There are two registration times, indicating when the valid-time start and the valid-time end were recorded. The single deletion time indicates when a tuple is logically deleted. Together, the registration times and the deletion time provide support for transaction time. A relation of tuples with these temporal attributes is termed a *Time Relation*.

Temporal selection is supported by the `TIME-VIEW` operator and a few simple, built-in temporal predicates. The `TIME-VIEW` operator produces a *snapshot relation*, i.e., a relation without timestamps [Snodgrass & Ahn 1985]. The operator accepts two parameters, an `E-TIME` and an `AS-OF` time. The syntax is given as follows (defaults are underlined).

$$\texttt{TIME-VIEW}\ \left[\texttt{E}-\texttt{TIME} = \left\{\begin{array}{c}\underline{\texttt{NOW}}\\ \texttt{CURRENT}\\ \text{<temporal constant>}\end{array}\right\}\right]\left[\texttt{AS}-\texttt{OF} = \left\{\begin{array}{c}\texttt{NOW}\\ \underline{\texttt{CURRENT}}\\ \text{<temporal constant>}\end{array}\right\}\right]$$

To compute the resulting snapshot relation, the argument relation is first (transaction) time-sliced `AS-OF` some time in the past. Then the intermediate relation is (valid) time-sliced as of the `E-TIME`. The result is the tuples that were effective at `E-TIME`, were registered before the `AS-OF` time, and not deleted before the `AS-OF` time. Because the resulting relation is a non-temporal relation, valid-time projection is not defined in TRM.

TRM provides a limited set of event-comparison operators which may be used in selection predicates.

```
WHERE <qualifier> (<field name>) <comp-operator> <temporal constant>
<qualifier> ::= E-START | E-END | R-START | R-END
<comp-operators> ::= < | <= | = | != | >= | >
```

No interval-comparison operators exist, and a temporal predicate is restricted to compare the start of or end of valid or registration times with temporal constants only; there is no way to compare two time variables. For example, a join that compares the start times of two relations can not be expressed in this language. Following is the sample query for Q1 expressed in TRM.

```
<temp-spec>       ::=<temp-period> <time-dimension>
<temp-period>     ::=AT [ PRESENT | <temporal const>]
                    |  DURING "("<temporal const> "-" <temporal const> ")"
                    |  BEFORE <temporal const>
                    |  AFTER <temporal const>
                    |  WHILE <selection expr>
                       DURING [ "(" -∞ "-" +∞")"
                                |  "("<temporal const> "-" <temporal const>")" ]
<time-dimension> ::=ALONG RT | ALONG <tsa>
```

Figure 3: BNF Syntax for Constructs in TOSQL Related to Timeslice

```
SELECT NAME
FROM EMPLOYEE
WHERE E-START(NAME) < 12.31.1990 AND E-END(NAME) > 1.1.1990
```

The temporal selection predicate in this query simply tests whether the valid times of employee tuples cover the year 1990, but a long and unclear predicate is necessary because interval comparison operators are not available.

2.4 TOSQL

TOSQL is based on tuple timestamped *transaction-time* relations [Ariav 1986]. In this model, tuples are atomatically assigned a *recording time*, abbreviated RT (i.e., a transaction time, the same as registration time in TRM). To obtain additional temporal support, it is possible to define so-called *Time-Related Attributes* (TRAs). Such attributes record time-varying data, for example the times when equipment malfunctions occurred. TRAs that satisfy the *finality property* are termed *Timestamp Attributes* (TSAs). This property is satisfied if at any give time, there exists one and only one value set for each object in the relation. With TSAs, it is possible to record, the valid times of tuples as well as other times. When a TSA is introduced into a relation, its dimensionality is increased by one.

In TOSQL, timeslice may be specified in five ways, each of which are shown in Figure 3 . These operators serve as both temporal selection and projection operators for RT (the default) as well as existing TSAs. Temporal projection is computed as the intersection of the period of time given in the timeslice clause and the intervals of selected tuples; no alternatives exist for specifying the timestamps of selected tuples.

The sample query for Q1 is expressed as follows.

```
SELECT F1.NAME
FROM EMP F1
DURING [1.1.1990 - 12.31.1990]
```

Here, the operator `DURING` retrieves all of the tuples which are recorded in 1990, and the `SELECT` operator outputs the employees' names.

The functionality of these five clauses are similar. All of them, except the `WHILE` clause, can be simulated by the `DURING` clause. Even though the `WHILE` clause does take a selection predicate as

argument, it is different from the WHERE clause. It singles out periods of time in which tuples satisfy the selection predicate. It's effect may be simulated by a combination of a temporal predicate and the WHERE clause. This is illustrated in the following two queries.

Q2. List all the employees who work in the company at some time when Tom is in the toy department.

```
SELECT E.NAME
FROM EMPLOYEE E
WHILE E.NAME = 'Tom' AND E.DEPT = 'toy'
```

The same query, now formulated in TSQL (described in the next section), is expressed as follows.

```
SELECT E1.NAME
FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E2.NAME = 'Tom' AND E2.DEPT = 'toy'
WHEN E1.INTERVAL OVERLAP E2.INTERVAL
```

2.5 TSQL

Navathe's *temporal relational model* supports tuple timestamping for valid time by attaching two mandatory timestamp attributes, *Time-start* (Ts) and *Time-end* (Te) to every time-varying relational schema [Martin et al. 1987, Navathe & Ahmed 1987, Navathe & Ahmed 1989]. These timestamp attributes correspond to the lower and upper bounds of time intervals in which tuples are continuously valid.

Informally speaking, attributes are not allowed to have multiple values at any particular instant of time in relation instances in this model. As a result, the time invariant key (i.e., the primary key of the corresponding snapshot relation schema, abbreviated TIK) together with either Ts or Te defines a candidate key for a temporal relation. Next, *value-equivalent* tuples (i.e., tuples with identical non-timestamp attribute values) are required to be coalesced. Coalescing affects the facility for temporal ordering of tuples, and requiring it improves the utility of this facility.

Most temporal operations are supported in TSQL, as are the properties defined in Section 2.1. We will describe these operations next, starting with temporal predicates and continuing with temporal projection, temporal ordering, and timeslice.

In TSQL, a temporal selection predicate is specified in the when clause which is a temporal analogue to SQL's where clause. The clause consists of a new keyword WHEN followed by a temporal boolean expression (see Figure 4 for the BNF syntax) which in turn consists of temporal predicates and logical operators.

The predicates consist of temporal expressions, which return intervals or events, and temporal comparison operators. In temporal expressions, three postfix operators are used for referencing the timestamps of tuples. The event extraction operators .TIME-START and .TIME-END return the start of and end of time intervals, respectively. The interval extraction operator, .INTERVAL, returns the entire interval.

Eight comparison operators all of which apply to interval arguments are included, namely BEFORE, AFTER, EQUIVALENT, PRECEDES, FOLLOWS, OVERLAP, URING, and ADJACENT. Events are treated as de-

```
<time-slice-clause> ::=  TIME-SLICE <tem-const>
<when-claus>        ::=  WHEN <boolean>
<boolean>           ::=  <bool-term> | <boolean> OR <bool-term>
<bool-term>         ::=  <bool-fact> | <bool-term> AND <bool-fact>
<bool-fact>         ::=  [NOT] <bool-prim>
<bool-prim>         ::=  <predicate> | ( <boolean> )
<predicate>         ::=  <expr> <tem-comp-op> <expr>
<expr>              ::=  <tem-seq> <ts-var> [<bf-af> <tem-seq> BREAK]
                         | [<tem-seq>] <ts-var> | <tem-const>
<tem-const>         ::=  <t-term> | "[" <t-term> "," <t-term> "]"
<t-term>            ::=  <t-fac> <add-op> <number>
<t-fac>             ::=  <t-s-v> | NOW | <number> <t-s-f>
<tem-com-op>        ::=  OVERLAP | EQUIVALENT | FOLLOWS | BEFORE
                         | AFTER | ADJACENT | PRECEDES | DURING
<tem-seq>           ::=  FIRST | SECOND | THIRD | Nth | LAST
<ts-var>            ::=  <ts-attr> | <ts-attr-fid>
<ts-attr>           ::=  [<rel-name>"."] <time-st>
<ts-attr-fid>       ::=  [<rel-name>"."] <time-st>"."<t-s-f>
<time-st>           ::=  TIME-START | TIME-END
<bf-af>             ::=  BEFORE | AFTER
<t-s-v> : refer to value appearing under Ts(time start) or Te(time end).
<t-s-f> : temporal fields( e.g., year, month, day)
```

Figure 4: BNF of Temporal Selection in TSQL

generate intervals, so that the comparison operators also apply to events. Based on Allens's interval relationships, Table 1 provides an overview of the interval comparison operators of TSQL, HSQL, and TQUEL [Why's the sixth row there?]. The table shows that TSQL has operators that correspond to each of Allen's interval relationships, with the exception of start and finish (which are rarely used). The table also indicates that the three languages are all complete with respect to interval comparison. The semantics of the eight operators in TSQL is clear, so a TSQL temporal predicate is easier to understand and construct than predicates in the languages described in the previous three sections.

The sample query Q1 is expressed as follows in TSQL.

```
SELECT F1.NAME
FROM EMP F1
WHEN F1.INTERVAL CONTAINS "1990"
```

Valid-time projection is specified in the target list, i.e., in the `SELECT` clause, with the format of a relation quantifier name followed by `.TIME-START`, `.TIME-END`, or `.INTERVAL`. If there is more than one relation included in the query, the quantifier name is necessary in order to indicate which timestamp to retrieve; otherwise, the quantifier name can be omitted. An interval constructor, `INTER`, which returns the intersection of two intervals can be used in the target list when specifying the timestamps of the resulting relation. This is illustrated in query Q3 below.

Q3. For all employees that worked in the same department as Mike and worked throughout 1990, find the time when they worked with Mike.

```
SELECT F1.NAME, (F1 INTER F2).TIME-START, (F1 INTER F2).TIME-END FROM
EMPLOYEE F1, EMPLOYEE F2
```

Allen	TSQL	HSQL	TQUEL
a before b	a BEFORE b or b AFTER a	a PRECEDES b	a precede b and not (end of a equal begin of b)
a equals b	a EQUIVALENT b	a = b	a equal b
a meets b	a PRECEDES b or b FOLLOWS a	a MEETS b	end of a equal begin of b
a overlaps b	a OVERLAP b AND a.TIME-END PRECEDES b.TIME-END	a OVERLAP b AND a.TO < b.TO	a overlap b and end of a precedes end of b
a during b	a DURING b	b CONTAINS a	begin of a overlap b and end of a overlap b and not (a equal b)
a meets b or b meets a	a ADJACENT b	a ADJACENT b	end of a equal begin of b or end of b equal begin of a
a starts b	(a.TIME-START EQUIVALENT b.TIME-START) AND (a.TIME-END PRECEDES b.TIME-END)	a.FROM = b.FROM AND a.TO < b.TO	begin of a equal begin of b and end of a precede end of b
a finishes b	(a.TIME-START AFTER b.TIME-START) AND (a.TIME-END EQUIVALENT b.TIME-END)	a.FROM > b.FROM AND a.TO = b.TO	begin of a precede begin of b and end of a equal end of b

Table 1: Overview of Interval Comparison Operators in TSQL, HSQL, and TQUEL

```
WHERE F2.NAME = 'Mike' AND F1.DEPT = F2.DEPT
WHEN F1.INTERVAL OVERLAP 1990 AND F2.INTERVAL OVERLAP 1990
```

According to the BNF syntax, the `.INTERVAL` in the `WHEN` clause cannot be omitted for indicating a timestamp. However, the tuple variable name can represent the timestamp in the `SELECT` clause (not shown in this paper). This is an inconsistency in the design—there is no logical reason for this difference.

Temporal ordering is well supported in TSQL because users can retrieve any version of an entry in its global or local ordering (defined later). No other languages support both global and local ordering. The underlying data model ensures that in any relation instance, no two tuples with the same TIK values have overlapping intervals. Thus, tuples can be grouped based on their TIK value and then be assigned unique order numbers based on either the start or the end times of their intervals. This is termed the *global ordering*. It is posible for the intervals in a group to be non-consecutive, i.e., there may be *breaks*. Such breaks may also be ordered and assigned order numbers. The breaks introduce a sub-partitioning of the tuples in a group. The tuples in the sub-groups may also be ordered—this is termed the *local ordering*.

To specify global temporal ordering in TSQL (see Figure 4 for the syntax), an order number,e.g., `FIRST` or `SECOND`, is placed before an attribute variable. Considering local ordering, a desired sub-group can be specified by indicating a break number, e.g., `AFTER FIRST BREAK`, behind an attribute variable. Some examples describe the use of global and local temporal ordering.

Q4. Find the names of the employees who started in the toy department.

```
SELECT E.NAME
```

```
FROM EMPLOYEE E
WHERE FIRST(E.DEPT) = 'TOY'
```

Q5. For employees that joined the company a second time and received a salary exceeding $ 50,000, retrieve the name and the time when they joined.

```
SELECT E.NAME E.TIME-START
FROM EMPLOYEE E
WHERE FIRST(E.SALARY) > 50000 AFTER FIRST BREAK
```

Q6. Find the start time and salary of employees when they entered the toy department the first time.

```
SELECT E.NAME FIRST(E.SALARY) E.TIME-START
FROM EMPLOYEE E
WHERE E.DEPT = 'TOY'
```

Time-slice queries are expressed using a special clause, `TIME-SLICE`. In this clause, an interval, constructed using temporal constants enclosed within square brackets, or a time point, defined by a temporal constant, is specified. The temporal constant can be a constant expression thus allowing dynamic timeslices. This property is very useful for a fixed-length timeslice which moves along in the valid-time dimension. An example is given below.

Q7. On a monthly basis, list all employees who worked in the company the last year.

```
SELECT NAME
FROM EMPLOYEE
TIME-SLICE YEAR[NOW-1, NOW]
```

This query can be executed monthly without modification.

2.6 HSQL

As the previous data model, Sarda's HDBMS also supports valid time; however, unlike the data model mentioned previously, HDBMS represent valid time in a valid-time relation as a single non-atomic, implicit attribute [Sarda 1990A]. The query language of HDBMS is called HSQL[1]. The valid timestamps of tuples can be either intervals, in *state relations*, or events, in *event relations*. The interval comparison, event comparison, and interval operations are defined for operating on timestamps. The granularity is not fixed in HDBMS where users can define different granularities for each relation. Comparison of events in different granularities is defined in HDBMS by that a coarser event is converted to a interval of finer granularity, then the comparison of the interval and the event of finner granularity is executed. Other operations dealing with granularity, such as expand and coalesce, are also provided. Details of temporal predicates, valid-time projection and timeslice are described in the following.

The functionality of temporal predicates and timeslices in HSQL is similar to that of TSQL. The difference are mainly in syntax and temporal comparison operators. A simplified BNF is shown in Figure 5 (Since Sarda does not provide any full syntax in his articles, the BNF is derived by us and may not be accurate, but it should be close to HSQL as proposed in Sarda's paper).

[1] In another paper, Sarda gave this extension to SQL the name TSQL [Sarda 1990B]. We use HSQL because it was used in the most recent paper.

```
<timeslice-clause>     ::= FROMTIME <temporal const> [TOTIME <temporal const>]
<where-clause>         ::= WHERE <boolean>
<boolean>              ::= <bool-term> | <boolean> OR <bool-term>
<bool-term>            ::= <bool-factor> | <bool-term> AND <bool-factor>
<bool-factor>          ::= [NOT] <bool-prim>
<bool-prim>            ::= <regular pred> | <temporal pred> | ( <boolean> )
<temporal pred>        ::= <interval> <interval-comp-op> <interval>
                         | <instant> <inst-comp-op> <instant>
                         | <instant> IN <interval>
<interval>             ::= <interval-fac> <interval-op> <interval-factor>
                         | <inst-factor> ". ." <inst-factor>
<interval-op>          ::= + | *
<interval-fac>         ::= <interval-var> | (<interval>)
<interval-var>         ::= <rel-name>"."INTERVAL
<inst-factor>          ::= <temp-const> | <rel-name>"."FROM | <rel-name>"."TO
<interval-comp-op>     ::= PRECEDES | = | MEETS | OVERLAPS | CONTAINS | ADJACENT
<inst-comp-op>         ::= < | <= | = | <> | >= | >
```

Figure 5: Simplified BNF for HSQL Temporal Selection

Unlike TSQL, which provides a `when` clause for temporal predicates, HSQL simply adds temporal predicates to the `where` clause. Boolean expressions in the `where` clause are composed of temporal predicates and non-temporal predicates. The event extraction operators are .FROM and .TO which retrieve the start and end of an interval, respectively. The timestamp-referencing operators are .INTERVAL and .AT; the .INTERVAL represents the interval timestamp of a tuple in a state relation, and the .AT indicates the event time associate with a tuple of a event relation. Another difference between temporal predicates in TSQL and HSQL is that HSQL supports two different set of comparison operators for interval time and event time. The arithmetic comparison operators are overloaded for event time. while these are the interval comparison operators: PRECEDES, =, MEETS, OVERLAPS, CONTAINS, ADJACENT. The only exception is the operator IN which tests if an event is contained in an interval. When formulating queries, users of HSQL must pay more attention than must TSQL users in order to choose the right comparison operators for the different temporal variables and constants. Although the set of interval comparison operators is smaller than that of TSQL, the expressive power is the same, see Table 1.

In order to explain how valid-time projection is done in HSQL, we introduce the interval constructors which are defined as follows.

- t_1 .. t_2: constructs a period of time from t_1 to t_2. It is null if $t_1 >= t_2$.
- p_1 + p_2: construct a period equals (min(p_1.FROM, p_2.FROM), max(p_1.END, p_2.END)) if p_1 and p_2 overlap; otherwise, it returns a null interval.
- p_1 * p_2: return the intersection of p_1 and p_2. If there is no intersection, a null interval is returned.

The valid-time projection is explicitly specified the timestamps in the select clause. The timestamps are either constructed by the interval constructors with timestamps of selected tuples or just the timestamps of selected tuples. If no timestamp is specified in the `select` clause then the result is a snapshot relation. For example, the query "list the employees in toy the department in 1990" can be expressed as follows.

Q8. List the employees in toy department in 1990.

```
FROMTIME 1.1.90 TOTIME 12.31.90
SELECT R.NAME, R.PERIOD * (1.1.90 ..  12.31.90)
FROM EMPLOYEE R
WHERE R.DPET = 'TOY'
```

This is also an example of timeslice. If TOTIME is omited, the default time, NOW, is taken. Sample formulations of the query **Q1** are given below. Note that there are two ways to write the query. One uses a temporal predicate; the other uses the timeslice operation.

```
SELECT F1.NAME
FROM EMP F1
WHERE F1 OVERLAP "1990"
```

```
FROMTIME 1:1:1990 TOTIME 12:31:1990
SELECT F1.NAME
FROM EMP F1
```

2.7 TempSQL

Gadia's TempSQL is based on a N1NF relational temporal data model which is attribute-value timestamped [Gadia & Vaishnav 1985, Gadia 1988, Gadia 1992]. A tuple may have more than one value (timestamped) for each attribute, but the union of the timestamps in each attribute must be the same throughout the entire tuple, resulting in a homogeneous temporal relation. In their data model, they group a history of an entry into a tuple, instead of storing the history of an entry in multiple tuples as is done in 1NF data models. The following is an example of an attribute timestamped relation.

NAME	SALARY	DEPT
[11,60] John	[11,49] 15K [50,54] 20K [55,60] 25K	[11,44] Toys [45,60] Shoes
[0,44] ∪ [50,NOW] Mary	[0,44] ∪ [50,NOW] 25K	[0,44] ∪ [50,NOW] Credit

In TempSQL, valid-time selection is heavily dependent on *temporal expressions* which are specified in the WHILE clause; see Figure 6 for a BNF. Temporal expressions are of the form, **[** A **]**, **[** r **]**, **[** AθB **]**, and **[** A θ b **]** where A and B are attributes, b is an attribute value and r is a relation. Thus the timestamp referencing is specified as a temporal expression. A temporal expression returns a *temporal element* [Gadia 1988] which is the finite union of intervals where the temporal expression is true. More complex temporal expressions are formed using the operators $\cup$, $\cap$, and $-$. For example, **[** SALARY = 20K **]** is a temporal expression. If the expression is applied to John's tuple, it would return [50,54].

There are two ways to specify temporal selection in TempSQL, depending on whether the time domain of the resulting relation is restricted to time elements specified in the WHILE clause. For example, the following query is interpreted as "list the employees when they worked in the toy department".

```
SELECT *
WHILE [[ DEPT = TOYS ]]
```

<select stmt> ::= <select clause> <while clause> <from clause> <where clause>
<select clause> ::= **select** <target list> [:<attr list>]
<while clause> ::= **while** <temporal expr>
<from clause> ::= **from** <relation list>
<where clause> ::= **where** <bool expr>
<relation list> ::= <relation name> [:<attr list>] [, <relation list>]}*
<temporal expr> ::= <temporal value>
| ¬ <temporal value>
| <temporal value> <set oper> <temporal value>
| <temporal constant>
<set oper> ::= ∩ | ∪
<temporal value> ::= **[** <attr name> **]**
| **[** <predicate> **]**
| **[** <select stmt> **]**
| **[exists** <select stmt> **]**
| **[** <attribute> **in** <select stmt> **]**
| **[** <attribute> **not in** <select stmt> **]**
| **[** <attribute> <comparison oper> **any** <select stmt> **]**
| **[** <attribute> <comparison oper> **all** <select stmt> **]**
<bool expr> ::= <bool value>
| ¬ <bool value>
| <bool value> <bool oper> <bool value>
| "(" <bool value> ")"
<bool oper> ::= **and** | **or**
<bool value> ::= <predicate>
| <temporal value> = <temporal value>
| <temporal value> ≠ ∅
| <temporal value> ⊇ <temporal value>
<predicate> ::= <attribute> <comparison oper> <attribute>
| <attribute> <comparison oper> <value>
| <value> <comparison oper> <attribute>

Figure 6: BNF for Temporal Selection in TempSQL

```
FROM EMP
```

The result is as follows.

NAME	SALARY	DEPT
[11,44] John	[11,44] 15K	[11,44] Toys

A similar query, "list the employees who worked in the toy department", would list any one who had ever worked in the toy department.

```
SELECT *
FROM EMP
WHERE [[ DEPT = TOYS ]]≠ ∅
```

The result is as follows.

NAME	SALARY	DEPT
[11,60] John	[11,49] 15K [50,54] 20K [55,60] 25K	[11,44] Toys [45,60] Shoes

In the first query, the time domain of the result is restricted by the temporal expression, `[ DEPT = TOYS ]`. The restriction represents a valid-time projection, i.e., valid-time projection is defined by the `WHILE` clause. The timestamp of the result relation is the intersection of the attribute timestamp with the temporal expression. In the second query, there is no restriction on the time domain of the result, so the entire tuple which satisfies the `WHERE` clause is retrieved.

There is no special timeslice operator defined in TempSQL (likewise for temporal ordering). Event extraction could be specified by two functions `firstInstant` and `lastInstant`. But it is not clear whether or not event comparison operators are supported.

2.8 TQUEL

Snodgrass's TQUEL is based on a tuple timestamped, temporal data model supporting both transaction time and valid time. Unlike TRM, where the result of a query is a snapshot relation, the result of a TQUEL query is a valid-time relation, so the data model is consistent. Like HDBMS, TQUEL distinguish between event relations and interval relations ("state relation" in HDBMS); however, in TQUEL the temporal comparison operators are overloaded for both interval and event time.

TQUEL adds a new clause, `when`, for valid-time selection.

Although not an extension of SQL, the BNF of the `when` clause as shown in Figure 7 still gives a good illustration of how valid-time selection may be done syntactically in TSQL2. The temporal

```
<when clause>     ::=  when <temporal pred>
<temporal pred>   ::=  <ei-expression> precede <ei-expression>
                  |    <ei-expression> overlap <ei-expression>
                  |    <ei-expression> equal <ei-expression>
                  |    <temporal pred> and <temporal pred>
                  |    <temporal pred> or <temporal pred>
                  |    (<temporal pred>)
                  |    not <temporal pred>
<ei-expression>   ::=  <e-expression>
                  |    <i-expression>
<e-expression>    ::=  <event element>
                  |    begin of <ei-expression>
                  |    end of <ei-expression>
                  |    (<e-expression>)
<i-expression>    ::=  <interval element>
                  |    <ei-expression> overlap <ei-expression>
                  |    <ei-expression> extend <ei-expression>
                  |    (<i-expression>)
<valid-clause>    ::=  valid from <e-expression> to <e-expression>
                  |    valid at <e-expression>
```

Figure 7: BNF for Temporal Selection in TQUEL

predicates can be partitioned into two groups: the first is predicates that consist of interval expressions, event expressions, and the temporal predicate operators, `overlap, precede,` and `equal`, which are overloaded for both interval and event time comparison; the second group is the boolean expressions that consist of predicates of the first group and logical operators.

The timestamps of tuples are referenced by tuple-variable names and the event extraction operators, `begin of` and `end of`. There are also two interval constructors, `overlap` and `extend`, for constructing temporal expression. The `overlap` operator returns an intersection of time when both arguments are valid. On the other hand, the `extend` operator is more like temporal union, in that it returns the points in time when either of the arguments are valid. These operators can be specified in event expressions and interval expressions for constructing a new event or interval. The sample query **Q1** is expressed as follows .

```
range of f1 is Emp
retrieve into r(name = f1.name)
when f1 overlap "1990"
```

Although, the set of comparison operators has three operators only, together wither the functions of timestamp referencing and event extraction, the set has the same expressive power as the other languages (see Table 1). However, long temporal expressions in TQUEL may be used to express similar predicates in other languages. For example, in Table 1, the operator `during` in TSQL is expressed as the conjunction of three predicates in TQUEL.

The `valid` clause is used for specifying valid-time projection, which in TQUEL can be any period of time. If the derived relation is to be an event relation, the `valid at` <t1> variant specifies a single timestamp. The other variant of the valid clause, `valid from` <t1> to <t2>, specifies an interval timestamp and is used when the derived relation is to be an interval relation. Both of t1 and t2 can be derived from event expressions given in previous paragraphs. The query **Q3** is expressed in the following

query which contains timestamp referencing, event extraction, interval constructor, valid clause, and temporal predicate.

```
range of e1 is employee
range of e2 is employee
retrieve into r(name= e1.name)
valid from begin of (e1 overlap e2) to end of (e1 overlap e2)
where e2.name = "Mike" and e1.dept = e2.dept
when e1 overlap 1990 and e2 overlap 1990
```

2.9 HQUEL

Tansel's HQUEL is based on a valid-time relational data model where the attributes are timestamped, leading to non-first normal form relations (N1NF) [Tansel & Arkun 1986]. The data model has four different kind of attributes which are elementary (atomic), set-valued, triplet-valued , and set triplet-valued attributes. The elementary and set-valued attributes are non-time-varying attributes; the other two are time-varying attributes. The values of triplet-valued attributes are triplets containing an element from the attribute's value domain and the boundary points of the interval of existence, while the value of set triplet-value attribute is a set of such triplets. The relation resulting from a query retains these four kinds of attributes depending on how the query is specified.

Because of the varieties of attributes, HQUEL supports methods for referencing members of a set, for indicating the timestamps and value of a triplet, and for comparing attribute values. Set members are referenced by a new range variable which ranges over the elements of a set. We describe this by an example. Let t be a regular range variable, as in QUEL, and `*A` be a set-valued attribute. `Range of s is t.*A` declares that s ranges over the elements of attribute `A` for each tuple in relation `t`. Assuming that `$B` is a triplet-valued attribute then `t.$B(V)`, `t.$B(L)`, `t.$B(U)`, and `t.$B(T)` represent the value, lower-bound(time start), upper-bound(time end) and interval of a triplet valued attribute, `t.$B`.

The new comparison operators include set comparison operators ($\rho \in \{=, \neq, \supset, \supseteq\}$), a set membership operator ($\in$), and temporal comparison operators. A interval comparison expression is of the form $x\theta y\rho z$ where x, y and z are time intervals, and $\theta \in \{\cap, \cup, \text{-}\}$. For example, "`t.$B(T)` $\cap$ `t.$C(T)` $\neq \emptyset$" means that the intersection of the time of t[$B] (denotes the set of triplet-valued attributes B in relation t) and the time of t[$C] is not empty, or simply, intervals of B and C overlap. This is not intuitive to users, since time is treated as a set of discrete points instead of a contiguous time line. Not only that, the interval comparison expression does not have the same expressive power as Allen's definition; for example, there is no way to express "before" in HQUEL.

For event comparison, the arithmetic comparison operators ($<, <=, =, \neq, >=, >$) are used to express an event-time predicate which contains event-time variables or temporal constants. For example, `t.$B(L)` $>$ `1/91` means the time of triplet-valued attribute B start after January 1991. Although, the interval comparison operators are not complete in HQUEL, the event-time comparison operators and timestamp retrieval operators have the same expressive power as other languages in Table 1. The sample query **Q1** is expressed as follows, assuming that `position` is a set triplet-valued attribute.

```
range of f1 is Emp
range of p1 is f1.*$position
retrieve into r(f1.name)
where p1.$(T) ∩ "1990" != ∅
```

In HQUEL, the valid-time projection is defined explicitly in the retrieve clause. If an attribute of a result relation is supposed to be a time-varying attribute, a triplet-valued attribute is specified in the `retrieve` clause. The set operators can also be used in valid-time projection for constructing timestamps in HQUEL. The following is an example.

Q9. What was Tom's position when he worked for the toy department, and when was it?

```
range of e is Emp
range of s is e.*$position
range of d is e.*$dept
retrieve into Tompos($Tpos = <s.$position(T) ∩ d.$dept, s.$position(V)>)
where e.ename = Tom and d.$dept(V) = Toy and s.$position(T) ∩ d.$dept
≠ ∅
```

2.10 HTQUEL

Gadia's HTQUEL is based on the same data model for TempSQL, but HTQUEL is an extension of QUEL. We will not discuss the data model again, but talk only about the language features. The function for timestamp referencing is `tdom` which takes an attribute as argument and returns a *temporal element*, a finite union of disjoint intervals. Because the timestamp of an attribute is a set of intervals, the functions for event extraction are applied on temporal elements. `Firstinstant(v)` and `Lastinstant(v)` return the two boundary time points of the temporal element, v. `Nextinstant(v,t)` takes an temporal element, v, and a instant, t, then returns an instant after t; `Previousinstant` is similar, but returns an instant before t. The functionalities of the functions, `Firstinterval, Lastinterval, Nextinterval, and Previousinterval`, are similar to the event extraction functions, but these four operations return intervals. HTQUEL supports timeslice via the `during` clause. Most of the other time-related operations are not defined clearly in the papers (e.g., such as event comparison, interval comparison, and temporal projection). We can thus not compare the expressive power with that of other languages. The sample query **Q1** is expressed as follows.

```
range of f1 is Emp
retrieve into r(name = f1.name)
during [1990]
```

2.11 Summary

In this section, we summarize the languages with respect to valid-time selection and projection, in two tables (Table 2 and Table 3), and we evaluate the query languages on ten properties (defined below). The first row in the tables contain the references to each language and the remaining rows each relate to a property. The first three properties, timestamp referencing, event extraction and event comparison, are essential to temporal predicates; the interval comparison and time slice improve syntactic convenience, but not the expressive power. An *event constructor* is a function which takes events as arguments and returns an event. An example of this is `first` which returns an oldest event from the arguments. With event constructors and interval constructors, we can write queries with advanced event or interval expressions. The remaining 3 properties are listed for comparison; the language that supports some of these properties is not necessarily better than other languages.

Language	TRM	TOSQL	TSQL	HSQL	TempSQL
Reference	[Ben-Zvi 1982]	[Ariav 1986]	[Navathe 1986] [Navathe 1987]	[Sarda 1990A] [Sarda 1990B]	[Gadia 1992]
Timestamp Referencing	*No*	*No*	*t*.INTERVAL *or* *tuple variable name* *(in* SELECT *clause)*	*t*.INTERVAL *t*.AT	[*relation name*]
Event Extraction	E-START(*attr*) E-END(*attr*) R-START(*attr*) R-END(*attr*)	*No*	*t*.TIME-START *t*.TIME-END	*t*.FROM *t*.TO	firstInstant[*u*] lastInstant[*u*]
Event time Comparison	<, =, >	*No*	*same as below*	<, =,>	*unclear*
Interval Comparison	*No*	*No*	BEFORE AFTER EQUIVALENT PRECEDES FOLLOWS OVERLAP DURING ADJACENT	PROCEDES = MEETS OVERLAPS CONTAINS ADJACENT	$\cap, \cup, \supseteq$
Event Constructors	*No*	*No*	*No*	*No*	*unclear*
Interval Constructors	*No*	*No*	INTER	t_1 .. t_2 * +	[*temp expr*]
timeslice Construct	TIME-VIEW	AT DURING BEFORE AFTER WHILE	TIME-SLICE	FROMTIME *t*1 TOTIME *t*2	*None*
Temporal Ordering	T-FIRST T-LAST	*No*	FIRST SECOND THIRD NTH LAST	FIRST LAST	*No*
New clause or Operators for Valid-time Selection	*No*	*No*	WHEN	*No*	WHILE
Temporal Valid-time	*No*	*Mixed with timeslice operators*	*SELECT clause vs WHEN clause*	*SELECT clause vs WHERE clause*	*WHILE clause vs WHERE clause*

Table 2: Summary (Part 1)

Language	Legol 2.0	TQUEL	HQUEL	HTQUEL
Reference	[Jones 1979]	[Snodgrass 1985] [Snodgrass 1987]	[Tansel 1986]	[Gadia 1985]
Timestamp Referencing	*tuple variable*	*tuple variable*	`$`*attr*`(T)`	`tdom(attr)`

Event Extraction	`start of(`*exp*`)`
	`end of(`*exp*`)`
		`begin of` *exp*
		`end of` *exp*
			`$`*attr*`(L)`
			`$`*attr*`(U)`
				`Firstinstant(`ν`)`
				`Lastinstant(`ν`)`
				`Nextinstant(`ν`,`i`)`
				`Previousinstant(`ν`,`i`)`

Event time Comparison	`<, =, >`	`precede` `equal`	`<, =, >`	*Unclear*

Interval Comparison	*Partial*	`precede`
		`equal`
		`overlap`
			Partial
			($=, \neq, \subset, \subseteq$)
				Unclear

Event Constructors	*No*	*No*	*No*	*No*

Interval Constructors	*No*	`overlap`
		`extend`
			No	`Firstinterval(`ν`)`
				`Lastinterval(`ν`)`
				`Nextinterval(`ν`,`i`)`
				`Previousinterval(`ν`,`i`)`

timeslice Construct	*None*	*None*	*None*	`During`

Temporal Ordering	`first`
	`last`
	`current`
	`past`
		No	*No*	*No*

New clause or Operators for Valid-time Selection	`while`
	`since`
	`until`
	`during`
	`while not`
	`or while`
	`union`
	`is`
	`is not`
		`when`	*No*	*No*

Valid-time Projection	*Mixed with temporal selection*	*valid clause vs when clause*	*From target list vs where clause*	*Unclear*

Table 3: Summary (Part 2)

In the summary table, we show the keywords of a language, if the language supports the property. Otherwise, "*No*" denotes not supporting a property in a certain language, and "*Unclear*" denotes that the original papers were unclear with respect to the property. In the previous subsections, we have discussed how each aspect is done in each language. We now describe these properties briefly and point out some interesting facts.

- *timestamp referencing*: How can timestamps of tuples be referenced? Generally, there are two ways: by an operator or through the tuple variable name. TSQL and HSQL use operators; Legol 2.0 and TQUEL use tuple-variable names to denote the timestamp of a tuple.
- *Event extraction*: What are the functions or operators for extracting the delimiters of a interval. The operators fall into two categories: prefix operators or postfix operators. The prefix operators are more like function calls that take interval expression as arguments. Postfix operators look like attribute and cannot operate on expressions, which limits their power. Because the data model of HQUEL and HTQUEL are N1NF, they both use special ways to extract events.
- *Event time comparison*: What are the operators for comparing events? All of the operators in these languages are infix and take event time constants and event time expressions as arguments. Half of the languages borrow the arithmetic comparison operators while TSQL and TQUEL use keywords as operators.
- *Interval comparison*: Are there operators for comparing intervals? Three languages—TSQL, HSQL, and TQUEL—support sets of helpful interval comparison operators, while other languages provide either no support or partial support.
- *Event constructor*: Is there any constructor that takes event time as arguments and returns an event? None of the nine languages support event constructors; however, we believe that event constructors are useful. This is the reason why event constructors are included in the summary.
- *Interval constructor*: Is there any constructor that operates on intervals or events and returns an interval? The common interval constructors are intersect and union(extend), but the definition and syntax may vary from language to language.
- *timeslice construct*: Is a timeslice construct supported in the language? Generally, the timeslice operation can be replaced by a simple temporal predicate (shown in next section). "*None*" denotes that no special construct for timeslice exists. Otherwise, we give the keywords.
- *Temporal ordering*: Is there a way to retrieve a tuple from a group according to its temporal ordering in the group? Most query languages provide functions for the first and last in temporal order, except TSQL which provides a set of functions for retrieving any version of an entity.
- *New clause or operators for temporal selection*: Is a new clause defined for temporal predicates in the language? Most of the languages do not introduce a new clause for temporal predicates—TQUEL, TSQL, and Legol 2.0 are exceptions. Legol 2.0 is a procedural query language which needs many operators to accomplish the desired functionalities.
- *Valid-time projection*: How are the timestamps defined for the resulting relation? There are three possibilities: the valid-time projection can be defined in the target list, it can be defined in a new clause, or it can be mixed with valid-time selection operators. TSQL, HSQL, and HQUEL employ the target list, and TQUEL uses the `valid` clause. Both TOSQL and Legol 2.0 mix valid-time projection with valid-time selection.

Acknowledgements

This work was supported in part by NSF grants ISI-8902707 and ISI-9302244, IBM contract #1124 and the AT&T Foundation. Christian S. Jensen was in addition supported in part by the Danish Natural Science Research Council, grants no. 11–1089–1 SE and no. 11–0061–1 SE.

3 Bibliography

[Allen 1983] Allen, J.F. "Maintaining Knowledge about Temporal Intervals." *Communications of the Association of Computing Machinery*, 26, No. 11, Nov. 1983, pp. 832–843.

[Ariav 1986] Ariav, G. "A Temporally Oriented Data Model." *ACM Transactions on Database Systems*, 11, No. 4, Dec. 1986, pp. 499–527.

[Ben-Zvi 1982] Ben-Zvi, J. "The Time Relational Model." PhD. Dissertation. Computer Science Department, UCLA, 1982.

[Chamberlain et al. 1976] Chamberlain, D. D. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control." *IBM J.*, 20, No. 6 (1976), pp. 560–575.

[Gadia & Vaishnav 1985] Gadia, S.K. and J.H. Vaishnav. "A Query Language for a Homogeneous Temporal Database," in *Proceedings of the ACM Symposium on Principles of Database Systems*. Mar. 1985, pp. 51–56.

[Gadia 1988] Gadia, S.K. "A Homogeneous Relational Model and Query Languages for Temporal Databases." *ACM Transactions on Database Systems*, 13, No. 4, Dec. 1988, pp. 418–448.

[Gadia 1992] Gadia, Shashi K. "A Seamless generic extension of SQL for querying temporal data." preliminary. Computer Science Department, Iowa State University. Mar. 1992.

[Held et al. 1975] Held, G.D., M. Stonebraker and E. Wong. "INGRES–A Relational Data Base Management System," in *Proceedings of the AFIPS National Computer Conference*. Anaheim, CA: AFIPS Press, May 1975, pp. 409–416.

[Jones et al. 1979] Jones, S., P. Mason and R. Stamper. "LEGOL 2.0: A Relational Specification Language for Complex Rules." *Information Systems*, 4, No. 4, Nov. 1979, pp. 293–305.

[Lorentzos 1988] Lorentzos, N.A. "A Formal Extension of the Relational Model for the Representation and Manipulation of Generic Intervals." Ph.D thesis, Birkbeck College, University of London, 1988.

[Lorentzos & Johnson 1988] Lorentzos, N.A. and R.G. Johnson. "Extending relational algebra to manipulate temporal data." *Information Systems*, 13, No. 3, pp. 289–296.

[Martin et al. 1987] Martin, N.G., S.B. Navathe and R. Ahmed. "Dealing with Temporal Schema Anomalies in History Databases," in *Proceedings of the Conference on Very Large Databases*. Ed. P. Hammersley. Brighton, England: Sep. 1987, pp. 177–184.

[Navathe & Ahmed 1987] Navathe, S. B. and R. Ahmed. "TSQL-A Language Interface for History Databases," in *Proceedings of the Conference on Temporal Aspects in Information Systems*. AFCET. France: May 1987, pp. 113–128.

[Navathe & Ahmed 1989] Navathe, S. B. and R. Ahmed. "A Temporal Relational Model and a Query Language." *Information Sciences*, 49 (1989), pp. 147–175.

[Sarda 1990A] Sarda, N. "Extensions to SQL for Historical Databases." *IEEE Transactions on Knowledge and Data Engineering*, 2, No. 2, June 1990, pp. 220–230.

[Sarda 1990B] Sarda, N. "Algebra and Query Language for a Historical Data Model." *The Computer Journal*, 33, No. 1, Feb. 1990, pp. 11–18.

[Sarda 1990C] Sarda, N. "Time-rollback using Logs in Historical Databases." Technical Report. Indian Institute of Technology. June 1990.

[Sarda 1990D] Sarda, N. "Design of a Historical Database Management System," in *Proceedings of the CSI Indore Chapter Conference (invited paper)*. Aug. 1990.

[Snodgrass & Ahn 1985] Snodgrass, R. and I. Ahn. "A Taxonomy of Time in Databases," in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Ed. S. Navathe. Association for Computing Machinery. Austin, TX: May 1985, pp. 236–246.

[Snodgrass 1987] Snodgrass, R. "The Temporal Query Language TQuel." *ACM Transactions on Database Systems*, 12, No. 2, June 1987, pp. 247–298.

[Soo & Snodgrass 1992] Soo, M. and R. Snodgrass. "Mixed Calendar Query Language Support for Temporal Constants." TempIS Technical Report 29. Computer Science Department, University of Arizona. October 30, 1991 1992.

[Tansel & Arkun 1986] Tansel, A.U. and M.E. Arkun. "HQuel, A Query Language for Historical Relational Databases," in *Proceedings of the Third International Workshop on Statistical and Scientific Databases*. July 1986.