WLG-015

ISO

International Organization for Standardization

ISO/IEC JTC 1/SC 32

Data Management and Interchange

WG3 Database Languages

Secretariat: USA (ANSI)

Title: Function comparison of History proposals including SQL/Temporal, TSQL2

Source: Japan

Status: Discussion paper

Author: Takashi Kotera,, Kenji Suzuki, Kohsaku Yamahira, Kohji Shibano, Shojiro Tanaka, Takaaki Shiratori, Takayuki Hayakawa, Tomoyuki Kajino, Minoru Inui, Takashi Ishizaka, Yoshiyuki Ishii

Abstract: This paper gives function comparison among SQL/MM History, SQL/Temporal and TSQL2.

References:

[History] WG4:WLG-005 (ISO/IEC Working Draft 13249-7) Information technology — Database languages — SQL Multimedia and Application Packages — Part 7: History, October, 2005

[Temporal] WG3:YYJ-007 = H2-2001-144, (ISO-ANSI Working Draft) Temporal (SQL/Temporal), June, 2001

[TSQL2] TSQL2 Language Specifications, September, 1994

Discussion:

We make function comparison among SQL/MM History, SQL/Temporal and TSQL2. We show the result of this comparison as the attached table. For functions of [Temporal] and [TSQL2] that are not provided by [History] we give descriptions that rewrite those functions with SQL. We show the descriptions as follows. Each number following an asterisk corresponds to that marked on the attached table.

*1 DATE type

When DATE value is used in an application using SQL/MM History, an application can use HS_History type with converting DATE type value to TIMESTAMP value. To convert DATE value the application can explicitly specify TIMESTAMP value with the time fields whose values are zero, or can use CAST specification.

1) Explicit timestamp value

```
SELECT … FROM TABLE(HS_TBL_emp::HS_HistoryTable())
  WHERE HS_Hist.HS_Contains(TIMESTAMP'2005-04-01 00:00:00')
```

2) CAST specification

```
SELECT CAST(HS_History_BeginTime AS DATE)
  FROM HS_History::HS_Period('Title') WHERE EmpID = 1 AND Title = 'Professor'
```

*2 Time zone

When TIMESTAMP item without time zone is set to TIMESTAMP value with time zone, the value is implicitly converted. When TIMESTAMP item with time zone is set to a local TIMESTAMP value without time zone, the value is implicitly converted if explicit time zone is specified on a datetime expression.

1) Set HS_History_BeginTime attribute to the converted value from TIMESTAMP value with time zone

```
SELECT * FROM VALUES(
  HS_TYPE_emp::HS_HistoryBeginTime
    ( 1, TIMESTAMP'2005-03-31 16:00:00+08:00'))
```

2) Get HS_History_BeginTime attribute as TIMESTAMP value with time zone

```
SELECT MIN(HS_Hist.HS_HistoryBeginTime AT TIME ZONE
                              INTERVAL'08:00' HOUR TO MINUTE
  FROM TABLE(HS_TBL_emp::HS_HistoryTable()))
  WHERE EmpID = 1 AND Title = 'Professor'
```

*3 Period with the element type of exact numeric

The codes described with SQL depend on meaning on an application for a numeric value. Now we think of the case that an exact numeric value expresses the number of past years from 1988.

1) Update

```
SELECT * FROM VALUES(
  HS_TYPE_emp::HS_HistoryBeginTime
    ( 1, TIMESTAMP'1988-01-01 00:00:00' + CAST(17 AS INTERVAL YEAR))
```

2) Retrieval

```
SELECT EXTRACT(YEAR FROM HS_Hist.HS_HistoryBeginTime) - 1998
  FROM TABLE(HS_TBL_emp::HS_HistoryTable()) …
```

*4 Addition and substraction of period type and interval type

1) Addition

```
NEW HS_History(period-value.HS_HistoryBeginTime + interval-value ,
                    period-value.HS_HistoryEndTime + interval-value)
```

If the ending value in the second argument is null, then the result is null.

2) Subtraction

```
NEW HS_History(period-value.HS_HistoryBeginTime - interval-value ,
                    period-value.HS_HistoryEndTime - interval-value)
```

If the ending value in the second argument is null, then the result is null.

*5 WEIGHTED option in set function

The value returned by a set function with WEIGHTED option specified in TSQL2 can be got by calculating the number of granules. We think of the following query that calculates the sum of the salary over each period of the history row for each EmpID

where the granule of HS_Hist is MONTH.

```
SELECT EmpID, SUM(WEIGHTED Salary)
  FROM TABLE(HS_TBL_emp::HS_HistoryTable())
  GROUP BY EmpID, HS_Hist
```

This query is equivalent to the following query.

```
SELECT EmpID, SUM((HS_Hist.MonthInterval() + 1) * Salary)
  FROM TABLE(HS_TBL_emp::HS_HistoryTable())
  GROUP BY EmpID, HS_Hist
```

In the above query we assume the following user-defined ordering for grouping by HS_Hist column.

```
CREATE ORDERING FOR HS_History
  ORDER FULL BY MAP WITH METHOD HS_HistoryBeginTime()
```

*6 Set function RISING

RISING function specified in TSQL2 calculates the maximum period such as the argument of the function increases monotonically. The following query returns the maximum period such as the Salary increases monotonically for each EmpID.

```
SELECT EmpID, RISING(Salary) AS HS_Hist
  FROM TABLE(HS_TBL_emp::HS_HistoryTable())
  GROUP BY EmpID
```

This query is equivalent to the following query.

```
WITH RECURSIVE
  HTBL(EmpID, BTime, ETime)
  AS(SELECT HT1.EmpID,
            HT1.HS_Hist.HS_HistoryBeginTime,
            HT1.HS_Hist.HS_HistoryEndTime
     FROM TABLE(HS_TBL_emp::HS_HistoryTable()) AS HT1
     WHERE NOT EXISTS(SELECT *
```

```
                                    FROM TABLE(HS_TBL_emp::HS_HistoryTable())
                                        AS HT2
                                    WHERE HT2.EmpD = HT1.EmpID
                                        AND HT2.HS_Hist.HS_HistoryEndTime
                                            = HT1.HS_Hist.HS_HistoryBeginTime
                                        AND HT2.Salary <= HT1.Salary)
        UNION ALL
        SELECT HT1.EmpID,
                HT2.HS_Hist.HS_HistoryBeginTime,
                HT1.HS_Hist.HS_HistoryEndTime
            FROM TABLE(HS_TBL_emp::HS_HistoryTable()) AS HT1, HTBL AS HT2
            WHERE HT2.EmpD = HT1.EmpID
                AND HT2.HS_Hist.HS_HistoryEndTime
                    = HT1.HS_Hist.HS_HistoryBeginTime
                AND HT2.Salary <= HT1.Salary)
SELECT EmpID, NEW HS_History(BTime, MAX(ETime)) AS HS_Hist
    FROM HTBL AS HT1
    WHERE (Etime – BTime) YEAR TO MONTH
            = (SELECT MAX((Etime – BTime) YEAR TO MONTH
                FROM HTBL AS HT2 WHERE HT2.EmpD HT1.EmpID)
```

*7 Proximity function

The codes described with SQL depend on the sort of granule. We think of SQL expression for the granule of a day.

1) NEXT function

datetime-value + INTERVAL'1'DAY

2) PRIOR function

datetime-value - INTERVAL'1'DAY

*8 Next value of the last value

The codes described with SQL depend on the sort of granule. We think of SQL expression for the granule of a day.

period-value.HS_HistoryEndTime + INTERVAL'1'DAY

If the last value is null, the result is null because the next value is unknown.

*9 Interval qualifier other than MONTH and DAY
The following expression gives the value on the specified interval-qualifier.

(COALESCE(period-value.HS_HistoryEndTime, CURRENT_TIMESTAMP)
 - period-value.HS_HistoryBeginTime) interval-qualifier

*10 Period constructor with arguments of upper or lower bound

1) PERIOD(beginning-datetime-value, ending-datetime-value]

NEW HS_History(beginning-datetime-value - INTERVAL'1'DAY,
                ending-datetime-value + INTERVAL'1'DAY)

2) PERIOD(beginning-datetime-value, ending-datetime-value)

NEW HS_History(beginning-datetime-value - INTERVAL'1'DAY,
                ending-datetime-value)

3) PERIOD[beginning-datetime-value, ending-datetime-value]

NEW HS_History(beginning-datetime-value,
                ending-datetime-value + INTERVAL'1'DAY)

*11 Cast between HS_History type and character string type

1) Cast from HS_History type to character string type

CAST(period-value.HS_HistoryBeginTime AS CHAR(19)) || ','
   CAST(COALESCE(period-value.HS_HistoryEndTime, CURRENT_TIMESTAMP)
                                                AS CHAR(19))

If the following user-defined cast is defined, then the above cast is automatically

performed.

```
CREATE CAST(HS_History AS CHAR(39))
  WITH FUNCTION His_TO_CHAR(HS_History)

CREATE FUNCTION His_TO_CHAR(Hist HS_History)
  RETURNS CHAR(39)
  BEGIN
    RETURN
     CAST(Hist. HS_HistoryBeginTime AS CHAR(19)) || ','
     CAST(COALESCE(Hist.HS_HistoryEndTime, CURRENT_TIMESTAMP)
                                                    AS CHAR(19))
  END
```

2) Cast from character string type to HS_History type

```
    NEW HS_History(CAST(SUBSTRING(CHis FROM 1 FOR 19) AS TIMESTAMP),
                   CAST(SUBSTRING(CHis FROM 21 FOR 19) AS TIMESTAMP)
```

If the following user-defined cast is defined, then the above cast is automatically performed.

```
CREATE CAST(CHAR(39) AS HS_History)
  WITH FUNCTION CHAR_TO_His(CHAR(39))

CREATE FUNCTION CHAR_TO_His( CHis CHAR(39))
  RETURNS HS_History
  BEGIN
    DECALRE BTime TIMESTAMP;
    DECALRE ETime TIMESTAMP;
    SET BTime = CAST(SUBSTRING(CHis FROM 1 FOR 19) AS TIMESTAMP);
    SET ETime = CAST(SUBSTRING(CHis FROM 21 FOR 19) AS TIMESTAMP);
    RETURN NEW HS_History(BTime, ETime);
    END
```

*12 Cast between HS_History type and DATE type

1) Cast from HS_History type to DATE type

```
CREATE CAST(HS_History AS DATE)
  WITH FUNCTION His_TO_DATE(HS_History)

CREATE FUNCTION His_TO_DATE(Hist HS_History)
  RETURNS DATE
  BEGIN
    IF Hist. HS_HistoryBeginTime = Hist. HS_HistoryEndTime THEN
      RETURN CAST(Hist. HS_HistoryBeginTime AS DATE);
    ELSE
      SIGNAL ‘xxyyy’;
    END IF;
  END
```

2) Cast from DATE type to HS_History type

```
NEW HS_History(CAST(date-value AS TIMESTAMP),
                    CAST(date-value AS TIMESTAMP))
```

If the following user-defined cast is defined, then the above cast is automatically performed.

```
CREATE CAST(CHAR(39) AS HS_History)
  WITH FUNCTION DATE_TO_His(DATE)

CREATE FUNCTION DATE_TO_His(DHis DATE)
  RETURNS DATE
  BEGIN
    DECALRE BTime TIMESTAMP;
    SET BTime = CAST(DHis AS TIMESTAMP);
    RETURN NEW HS_History(BTime, BTime);
    END
```

*13 Cast between HS_History type and TIMESTAMP type

1) Cast from HS_History type to TIMESTAMP type

```
CREATE CAST(HS_History AS TIMESTAMP)
  WITH FUNCTION His_TO_TIMESTAMP(HS_History)

CREATE FUNCTION His_TO_TIMESTAMP(Hist HS_History)
  RETURNS DATE
  BEGIN
    IF Hist. HS_HistoryBeginTime = Hist. HS_HistoryEndTime THEN
      RETURN Hist. HS_HistoryBeginTime;
    ELSE
      SIGNAL ‘xxyyy’;
    END IF;
  END
```

2) Cast from TIMESTAMP type to HS_History type

```
NEW HS_History(timestamp-value, timestamp-value)
```

If the following user-defined cast is defined, then the above cast is automatically performed.

```
CREATE CAST(CHAR(39) AS HS_History)
  WITH FUNCTION TIMESTAMP_TO_His(TIMESTAMP)

CREATE FUNCTION TIMESTAMP_TO_His(THis TIMESTAMP)
  RETURNS DATE
  BEGIN
    RETURN NEW HS_History(THis, THis);
  END
```

*14 Cast between HS_History types with different element types

It can be got by constructing period value for arguments cast from the attribute of HS_History value.

*15 NORMALIZE function

NORMALIZE function specified in SQL/Temporal merges periods in a period set that overlaps or meets each other and generates a new period set that each period is not connected each other. The following user-defined function returns the result same as NORMALIZE function.

```
CREATE FUNCTION NormalizePeriod(Hist HS_History MULTISET)
  RETURNS HS_History MULTISET
  BEGIN
    RETUEN MULTISET(
    WITH RECURSIVE
      HTBL(BTime, ETime)
      AS(SELECT DISTINCT
            PR.HS_Hist.HS_HistoryBeginTime,
            COALESCE(PR.HS_Hist.HS_HistoryEndTime,
CURRENT_TIMESTAMP)
          FROM UNNEST(Hist) AS HT(PR)),
      PHTBL(BTime, ETime)
      AS(SELECT HT1.BTime, MAX(HT1.ETime)
            FROM HTBL AS HT1
            WHERE NOT EXISTS(SELECT *
                                FROM HTBL AS HT2
                                WHERE HT2.BTime < HT1.BTime
                                    AND HT2.ETime >= HT1.BTime)
            GROUP BY HT1.BTime
        UNION ALL
        SELECT PT.BTime, MAX(HT1.ETime)
          FROM PHTBL AS PT, HTBL AS HT1
          WHERE PT.ETime < HT1.ETime
              AND PT.ETime >= HT1.BTime)
          GROUP BY HT1.BTime
     SELECT NEW HS_History(BTime, MAX(ETime)) AS HS_Hist
        FROM PHTBL GROUP BY BTime
    );
  END
```

*16 EXPAND function

EXPAND function specified in SQL/Temporal disjoints a period to period values such as each period has one element datetime value contained in the argument period. The codes described with SQL depend on the sort of granule. We think of the case of the granule of a day.

```
CREATE FUNCTION ExpandPeriod(Hist HS_History)
  RETURNS DATE MULTISET
  BEGIN
    DECLARE Rsv HS_History MULTISET DEFAULT(MULTISET());
    DECLARE MElm DATE;
    DECLARE EElm DATE;
    SET MElm = CAST(Hist.HS_HistoryBeginTime AS DATE);
    SET EElm = CAST(Hist.HS_HistoryEndTime AS DATE);
    WHILE(MElm <= EElm) DO
      SET Rsv = Rsv UNION MULTISET[MElm];
      SET MElm = MElm + INTERVAL'1' DAY;
    END WHILE;
    RETURN SET(Rsv);
  END;
```

*17 Comparison predicate

Comparison of period values can be substituted to the comparison of row values.

Comparison predicate of period-value1 comparison-operator period-value2

This conparison is equivalent to the following comparison.

```
(period-value1.HS_HistoryBeginTime, period-value1.HS_HistoryEndTime)
comparison-operator
(period-value2.HS_HistoryBeginTime, period-value2.HS_HistoryEndTime)
```

*18 VALID clause

If VALID clause specified in TSQL2 is specified in a query specification, rows whose valid times are equal to the specified datetime or whose periods overlaps specified period are retrieved. The following query retrieves rows whose valid times are within

the period from 2000-04-01 to 2001-03-31.

```
SELECT EmpID, EmpName, Title
  VALID INTERSECT PERIOD'[1/4/2000 – 31/3/2001]'
  FROM HS_TBL_emp
```

The following query returns the same result as the above query.

```
SELECT EmpID, EmpName, Title
  FROM TABLE(HS_TBL_emp::HS_HistoryTable())
  WHERE HS_Hist.HS_Overlaps(DATE'2000-04-01', DATE'2001-03-31')
```

*19 Normalized query specification

A row of a history table has only one period in SQL/MM History, so we think of the case such as one period per each row. We think of the following query using normalized query specification specified in SQL/Temporal where PC1 and PC2 are the columns whose data types are both HS_History types.

```
SELECT C1, C2, …PC1, PC2,… FROM TBL …
  NORMALIZE ON PC1, PC2,…
```

We can construct the equivalent query as the above using normalizePeriod function defined in *15 as follows:

```
WITH NMTBL
  AS(SELECT NormalizePeriod(COLLECT(PC1)) AS PC1,
            NormalizePeriod(COLLECT(PC2)) AS PC2,…
            C1,C2,…
       FROM TBL GROUP BY C1, C2,…)
SELECT C1, C2, …, PC1, PC2, …
  FROM NMTBL,
       UNNEST(NMTBL.PC1) AS PT1(PC1),
       UNNEST(NMTBL.PC2) AS PT2(PC2),
```

*20 EXPANDING clause on query expression

EXPANDING clause specified in SQL/Temporal merges results of two queries with a

period for each row. We think of the following query using EXPANDING clause where PC1 and PC2 are the columns whose data types are both HS_History types.

```
SELECT C1, C2, …,PC1, PC2,… FROM TBL1 …
UNION ALL EXPANDING(PC1, PC2,…)
SELECT C1, C2, …PC1, PC2,… FROM TBL2 …
```

We can construct the equivalent query as the above using normalizePeriod function defined in *15 and ExpandPeriod function defined in *16. The description of the query depend on the sort of granule. We assume the granule of a day and one granule is expressed by the period whose ending timestamp is after the interval by a day from beginning timestamp.

```
WITH LTBL
  AS(SELECT ExpandPeriod(PC1) AS PC1, ExpandPeriod(PC2) AS PC2,…
            C1,C2,…
     FROM (SELECT C1, C2, … FROM TBL1 …) AS LT),
RTBL
  AS(SELECT ExpandPeriod(PC1) AS PC1, ExpandPeriod(PC2) AS PC2,…
            C1,C2,…
     FROM (SELECT C1, C2, … FROM TBL2 …) AS RT),
STBL
  AS(SELECT C1, C2, L1.PC1, L2.PC2,…
     FROM LTBL,
          UNNEST(LTBL.PC1) L1(PC1), UNNEST(LTBL.PC2) L2(PC2),…
   UNION ALL
     SELECT C1, C2, R1.PC1, R2.PC2,…
     FROM RTBL,
          UNNEST(RTBL.PC1) R1(PC1), UNNEST(RTBL.PC2)
R2(PC2),…),
NMTBL
  AS(SELECT NormalizePeriod(
              COLLECT(HS_History(PC1,PC1+INTERVAL'1'DAY))) AS PC1,
            NormalizePeriod(
              COLLECT(HS_History(PC2,PC2+INTERVAL'1'DAY))) AS PC2,…
            C1,C2,…
```

```
        FROM STBL GROUP BY C1, C2,…)
    SELECT C1, C2, …, PC1, PC2, …
      FROM NMTBL,
            UNNEST(NMTBL.PC1) AS PT1(PC1),
            UNNEST(NMTBL.PC2) AS PT2(PC2),
            …
```

*21 Grouping based on time granule

1) Grouping by time granule

The following query described by TSQL2 syntax calculates a sum of Salary over a month for each EmpID.

```
SELECT EmpID, SUM(Salary) AS YSal
  FROM HS_TBL_emp AS HT
  GROUP BY EmpID, VALID(HT) USING YEAR
```

The equivalent query can be written as follows:

```
WITH RECURSIVE
  HTBL(EmpID, Sal, Yer, Mth, YM)
  AS(SELECT HT1.EmpID, HT1.Salary,
            EXTRACT(YEAR FROM HT1.HS_Hist.HS_HistoryBeginTime),
            EXTRACT(MONTH FROM HT1.HS_Hist.HS_HistoryBeginTime),
            CAST(SUBSTRING(
                    CAST(HT1.HS_Hist.HS_HistoryBeginTime AS CHAR(19))
                        FROM 1 FOR 7)
                || '-01' AS DATE)
      FROM TABLE(HS_TBL_emp::HS_HistoryTable()) AS HT1
      WHERE NOT EXISTS(SELECT *
                        FROM TABLE(HS_TBL_emp::HS_HistoryTable())
                            AS HT2
                        WHERE HT2.EmpD = HT1.EmpID
                          AND HT2.HS_Hist.HS_HistoryEndTime
                              = HT1.HS_Hist.HS_HistoryBeginTime
                          AND HT2.Salary <= HT1.Salary)
```

```
    UNION ALL
    SELECT HT1.EmpID, HT1.Salary,
            EXTRACT(YEAR FROM HT2.YM + INTERVAL'1'MONTH),
            EXTRACT(MONTH FROM HT2.YM + INTERVAL'1'MONTH),
            HT2.YM + INTERVAL'1'MONTH
        FROM TABLE(HS_TBL_emp::HS_HistoryTable()) AS HT1, HTBL AS HT2
        WHERE HT2.EmpD = HT1.EmpID
            AND HT1.HS_Hist.Contains(HT2.YM + INTERVAL'1'MONTH)
  )
SELECT EmpID, Yer, SUM(Salary) AS Salary
  FROM HTBL
  GROUP BY EmpID, Yer
```

2) Specify the range of aggregate

The following query described by TSQL2 syntax returns the number of changes to Salary in past five years for each EmpID.

```
SELECT EmpID, SUM(Salary) AS YSal
  FROM HS_TBL_emp AS HT
  GROUP BY EmpID, VALID(HT) USING INSTANT LEADING INTERVAL'5'YEAR
```

The equivalent query can be written as follows:

```
  SELECT HT1.EmpID, HT1.Salary,
    FROM TABLE(HS_TBL_emp::HS_Period('Salary'))
                AS HT1(EmpID, Sal, Hist)
   WHERE Hist. HS_HistoryEndTime
                    BETWEEN CURRENT_TIMESTAMP – INTERVAL'5'YEAR
   GROUP BY EmpID
```

- End of paper -