TECHNICAL SPECIFICATION

ISO/IEC TS 13249-7

First edition
2013-02-15

Information technology — Database languages — SQL multimedia and application packages —

Part 7: History

Technologies de l'information — Langages de bases de données — Multimédia SQL et paquetages d'application —

Partie 7: Historique

Reference number
ISO/IEC TS 13249-7:2013(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In other circumstances, particularly when there is an urgent market requirement for such documents, the joint technical committee may decide to publish an ISO/IEC Technical Specification (ISO/IEC TS), which represents an agreement between the members of the joint technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/IEC TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/IEC TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 13249-7 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

ISO/IEC 13249 consists of the following parts, under the general title *Information technology — Database languages — SQL multimedia and application packages*:

— *Part 1: Framework*

— *Part 2: Full-Text*

— *Part 3: Spatial*

— *Part 5: Still image*

— *Part 6: Data mining*

— *Part 7: History* [Technical Specification]

Introduction

The purpose of ISO/IEC 13249 is to define multimedia and application specific types and their associated routines using the user-defined features in ISO/IEC 9075.

ISO/IEC 13249 is based on the content of ISO/IEC International Standard Database Language (SQL).

The organization of this Technical Specification is as follows:

1) Clause 1, "Scope", specifies the scope of this Technical Specification.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this Technical Specification, constitute provisions of this Technical Specification.

3) Clause 3, "Terms, definitions, notations, and conventions", defines the definitions, concepts, notations and conventions used in this Technical Specification.

4) Clause 4, "Concepts", presents concepts used in the definition of this Technical Specification.

5) Clause 5, "History Procedures", defines the history associated routines.

6) Clause 6, "History Types", defines the user-defined types provided for the manipulation of history.

7) Clause 7, "SQL/MM History Information Schema" defines the SQL/MM History Information Schema.

8) Clause 8, "SQL/MM History Definition Schema" defines the SQL/MM History Definition Schema.

9) Clause 9, "Status Codes", defines the SQLSTATE codes used in this Technical Specification.

10) Clause 10, "Conformance", defines the criteria for conformance to this Technical Specification.

In the text of this Technical Specification, clauses begin a new page. Any resulting blank space is not significant.

The history user-defined types and routines defined in this Technical Specification adhere to the following:

a) A history user-defined type and routine are generic to history data handling. History user-defined types and routines provide the means to record changes to the rows of a persistent base table in an SQL database, so that applications using such a persistent base table shall be completely independent of whether there is any recording of changes. This means that, when changes are to be recorded, an application does not need to be modified and its behaviour remains the same.

b) History user-defined types and routines provide the means to query the recorded changes for such a table.

c) A history user-defined type does not redefine the database language SQL directly or in combination with another history data type.

The scope of this Technical Specification is limited to support for history when there are no changes to the definition of the tracked columns of a tracked table. The following operations are not supported in this Technical Specification:

a) DROP COLUMN operation to a tracked column of a tracked table.

b) ALTER COLUMN operation to a tracked column of a tracked table except changes of the default value.

The scope of this Technical Specification is limited to support for history when a tracked table has at least one unique constraint with NOT NULL that is not modified by any ALTER TABLE statements.

If a transaction does not have an isolation level that is SERIALIZABLE, the results in the recorded history are implementation-dependent.

Information technology — Database languages — SQL multimedia and application packages —

Part 7: History

1 Scope

The ISO/IEC 13249 series defines a number of packages of generic data types common to various kinds of data used in multimedia and application areas, to enable that data to be stored and manipulated in an SQL database.

This Technical Specification:

a) defines concepts specific to this Technical Specification;

b) defines history user-defined types and their associated routines.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9075-1:2008, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*

ISO/IEC 9075-2:2008, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*

ISO/IEC 9075-4:2008, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*

ISO/IEC 9075-11:2008, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*

ISO/IEC 13249-1:2007, *Information technology — Database languages — SQL multimedia and application packages — Part 1: Framework*

3 Terms and definitions, concepts, notations and conventions

3.1 Terms and definitions

3.1.1 Terms and definitions provided in ISO/IEC 9075-1

This Technical Specification makes use of the following terms defined in ISO/IEC 9075-1:

a) atomic

b) fully qualified of a name of some SQL object

c) identify

d) object (as in 'x object')

e) persistent

f) SQL-session

3.1.2 Terms and definitions provided in ISO/IEC 9075-2

This Technical Specification makes us of the following terms defined in ISO/IEC 9075-2:

a) distinct (of a pair of comparable values)

b) qual (of a pair of comparable values)

c) dentical (of a pair of values)

d) SQL parameter

e) structured type

f) variable-length

3.1.3 Terms and definition provided in this Technical Specification

For the purposes of this document, the terms and definitions given in ISO/IEC 13249-1 and the following apply:

3.1.3.1
contiguous periods
sequence of two or more periods, such that, for all 1<i<=n (where n is the number of periods), the begin time of the i-th period is greater than the begin time of the (i-1)-th period and is equal to the end time of the (i-1)-th period

3.1.3.2
history row
row in a history table

3.1.3.3
history row set
set of rows in a history table that represent all the changes to a single row of a tracked table

3.1.3.4
history table
table that represents values of the tracked columns of a tracked table and the period during which all the values in each row were present in the tracked table

3.1.3.5
identifier column (of a tracked table)
either the primary key or one of the set of columns defined as UNIQUE and NOT NULL

3.1.3.6
period
duration of time with a begin time and an end time

NOTE In this Technical Specification a period value is a half-open duration that includes the begin time but not the end time.

3.1.3.7
period normalization
operation that makes one or more contiguous periods into a single period

3.1.3.8
period-normalized table
table resulting from selecting one or more columns in a history table, including the column(s) corresponding to the unique constraint columns with NOT NULL of the tracked table, and applying period normalization to rows that are otherwise not distinct

NOTE Each row in a period-normalized table is formed from one or more rows of a history table with the same values in one or more specified columns that relate to a continuous period, which may either be a single period from one row or contiguous periods from many rows.

3.1.3.9
tracked column
column of a tracked table for which changes are to be recorded

NOTE The tracked columns of a tracked table shall include unique constraint columns with NOT NULL of that table.

3.1.3.10
tracked row
row in a tracked table

3.1.3.11
tracked table
persistent base table for which changes are to be recorded for one or more tracked columns

3.1.3.12
transaction timestamp
timestamp value that is within the duration of an SQL-transaction

NOTE This value is implementation-dependent, preferably corresponding to the end of an SQL-transaction

3.2 Concepts

3.2.1 Concepts taken from ISO/IEC 9075-1

This Technical Specification makes use of the following concepts defined in ISO/IEC 9075-1:

a) assertion

b) domain

c) primary key

d) query

e) role

f) SQL-client module

g) SQL-schema

h) SQL-server module

i) SQL-transaction

j) SQLSTATE

k) trigger

l) unique constraint

3.2.2 Concepts taken from ISO/IEC 9075-2

This Technical Specification makes use of the following concepts defined in ISO/IEC 9075-2:

a) applicable role

b) authorization identifier

c) default unqualified schema name

d) default catalog name

e) enabled authorization identifier

f) parameter

g) procedure

h) sequence generator

i) SQL-path

j) SQL-session context

k) transaction timestamp

3.2.3 Syntactic elements taken from ISO/IEC 9075-2

This Technical Specification makes use of the following syntactic elements (BNF non-terminal symbols) defined in ISO/IEC 9075-2:

a) <catalog name>

b) <column name>

c) <comma>

d) <constraint name>

e) <data type or domain name>

f) <delimited identifier body>

g) <delimited identifier>

h) <double quote>

i) <equals operator>

j) <identifier body>

k) <local or schema qualifier>

l) <qualified identifier>

m) <quote>

n) <regular identifier>

o) <rollback statement>

p) <schema definition>

q) <schema name list>

r) <schema name>

s) <space>

t) <table name>

u) <Unicode delimiter body>

v) <Unicode delimiter identifier>

w) <unqualified schema name>

3.2.4 Other concepts

For the purposes of this document, the concepts given in ISO/IEC 13249-1 apply.

3.2.5 Notations

3.2.6 Notations provided in ISO/IEC 13249-1

For the purposes of this document, the notations given in ISO/IEC 13249-1 apply.

3.2.7 Notations provided in ISO/IEC 13249-7

This Technical Specification uses the prefix 'HS_' for view, base table, user-defined type, attribute and SQL-invoked routine names.

This Technical Specification uses the following representation in a figure for a table that includes the column of HS_Hist of the structured type, HS_History.

<column name>	*<column name>*	···	*HS_Hist (HS_BeginTime, HS_EndTime)*
Column Value	*Column Value*	···	*(Attribute Value, Attribute Value)*
Column Value	*Column Value*	···	*(Attribute Value, Attribute Value)*

…	…	…	…

3.3 Conventions

For the purposes of this document, the conventions given in ISO/IEC 9075-4, ISO/IEC 9075-11 and ISO/IEC 13249-1 apply.

4 Concepts

4.1 Overview

This Technical Specification provides user-defined types and routines that enable a user to specify columns of a table to record all changes to these columns and to query recorded changes. The recorded changes include tracking information on inserts, deletes, and updates of those columns of the table.

This part does not specify the means by which recorded changes are maintained. However, the concept of a history table is introduced in this clause, and it has two roles. A history table allows the precise specification of the user-defined types and routines providing the required capabilities, and it determines the way in which the recorded changes are materialised for querying.

4.1.1 Tracked Table and History Table

A tracked table is a persistent base table for which any changes to the current values of specified tracked columns are to be recorded. A history table is a means of virtualising the recording of these changes, even though there is no requirement for it to exist as a persistent base table.

A history table consists of the column of a sequence number, the columns corresponding to all tracked columns of the tracked table and the column of the structured type for the period of a history row.

Example 1:

Tracked Table TT1

ID	Column_A	Column_B	Column_C

History Table for Tracked Table TT1

HS_SEQ	ID	Column_A	Column_B	HS_Hist (HS_BeginTime, HS_EndTime)

Example 2:

Tracked Table TT2

ID	Column_A	Column_B	Column_C	Column_D	Column_E	Column_F	Column_G

History Table for Tracked Table TT2

HS_SEQ	ID	Column_A	Column_B	Column_C	Column_D	HS_Hist (HS_BeginTime, HS_EndTime)

In example 1, the column ID is the unique constraint column of the tracked table TT1. The columns ID, Column_A and Column_B are tracked columns of the tracked table TT1.

In example 2, the column ID is the unique constraint column of the tracked table TT2. The columns ID, Column_A, Column_B, Column_C and Column_D are tracked columns of the tracked table TT2.

The value of a begin time or an end time is automatically set by the system when an insert, update, or delete operation is executed on the tracked table TT1 or the tracked table TT2.

A row in the history table, namely history row, represents the values of the columns (in example 1, ID, Column_A and Column_B, and in example 2, ID, Column_A, Column_B, Column_C and Column_D) that existed from begin time to end time. Once a begin/end time is set to a certain non-NULL timestamp value, the value will not be changed.

The value of the column HS_SEQ is a number which is generated by an external sequence generator that is defined as START WITH 1 INCREMENT BY 1. The value of the column HS_SEQ is set when a history row is inserted into a history table.

4.1.2 Concept of Transaction Timestamp

When an insert operation, an update operation or a delete operation is executed on a tracked table, the values of the tracked columns need to be recorded along with a begin time and possibly an end time for the history period. But if multiple DML operations are executed in the same SQL-transaction, these operations will occur at different times and so using CURRENT_TIMESTAMP the resulted timestamp values may vary and would not allow the multiple changes for a transaction to be related. This is an especially serious problem if more than one table is being tracked. Thus a single time for all changes within a transaction is required. This requirement is provided by the implementation-dependent transaction timestamp for an SQL-transaction.

The value of transaction timestamp is used as the value of a begin time or an end time of a history row (see Section 4.1.3 Operations on Tracked Table).

The value of a transaction timestamp is determined by invoking the HS_GetTransactionTimestamp function defined in Subclause 5.3.9.

4.1.3 Operations on Tracked Table

A history table is created automatically for a tracked table by invoking the HS_CreateHistory procedure provided in this Technical Specification.

When the HS_CreateHistory procedure is invoked, a history table is initialized as follows. A history table is created and all rows in the tracked table are inserted into the history table, and the begin times of all history rows are set to the value of transaction timestamp of the execution time of the HS_CreateHistory procedure.

When an insert operation on a tracked table is executed, a history row, which corresponds to the inserted row in the tracked table, is inserted into the history table, and the begin time is set to the value of transaction timestamp of the transaction that executes the insert operation.

When an update operation on a tracked table is executed, for each row that is changed:

1) the end time of the row which has the latest begin time in the history row set for that row is set to the value of transaction timestamp of the transaction that executes the update operation, and

2) a history row which has the new values of the tracked columns for that row and the value of the transaction timestamp as its begin time is inserted into the history table.

When a delete operation on a tracked table is executed, for each row that is deleted, the end time of the row which has the latest begin time in the history row set for that row is set to the value of transaction timestamp of the transaction that executes the delete operation.

4.1.4 Operations on time periods

In order to query a history table, an application needs a means to specify conditions for the begin time or the end time when it is interested in. Also an application needs a means to extract the value of the begin time or the end time or to execute time-related operations.

Predicates to specify time-related condition to the history table are as follows. *X* and *Y* represent half-open periods ; [*b1*:*e1*) and [*b2*:*e2*) respectively, where the left square bracket shows that the begin of period is inclusive and the right parenthesis shows that the end of period is not inclusive, and *b1* and *b2* are the begin times and *e1* and *e2* are the end times.

— *X* Overlaps *Y* is:

  — true if and only if *b1* < *e2* and *b2* < *e1* are both true.

  — false if and only if either of *b1* < *e2* or *b2* < *e1* is false.

— *X* Contains *Y* is:

  — true if and only if *b1* <= *b2* and *e2* <= *e1* are both true.

  — false if and only if either of *b1* <= *b2* or *e2* <= *e1* is false.

— *X* Meets *Y* is:

  — true if and only if either of *e1* = *b2* or *e2* = *b1* is true.

  — false if and only if *e1* = *b2* and *e2* = *b1* are both false.

— *X* Precedes *Y* is:

  — true if and only if *e1* < *b2* is true.

  — false if and only if *e1* < *b2* is false.

— *X* PrecedesOrMeets Y is:

  — true if and only if either of *X* Precedes *Y* or *e1* = *b2* is true.

  — false if and only if *X* Precedes *Y* and *e1* = *b2* are both false.

— *X* Equals *Y* is:

  — true if and only if *b1* = *b2* and *e1* = *e2* are both true.

  — false if and only if either of *b1* = *b2* or *e1* = *e2* is false.

— *X* Succeeds *Y* is:

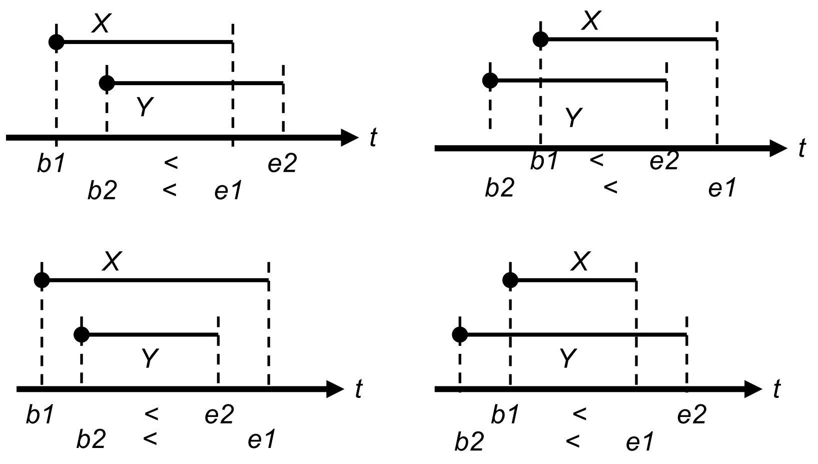
  — true if and only if *e2* < *b1* is true.

  — false if and only if *e2* < *b1* is false.

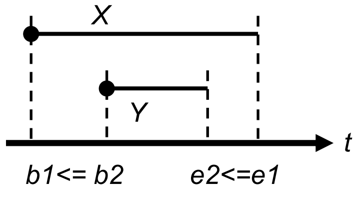
— *X* SucceedsOrMeets Y is:

  — true if and only if either of *X* Succeeds *Y* or *e2* = *b1* is true.

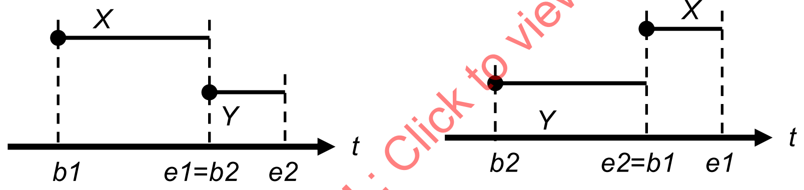
  — false if and only if *X* Succeeds *Y* and *e2* = *b1* are both false.



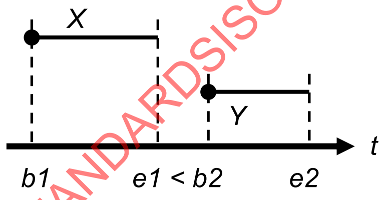
X Overlaps *Y*

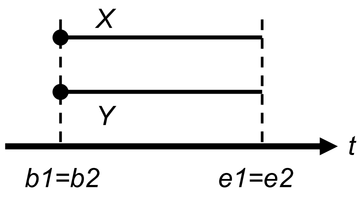


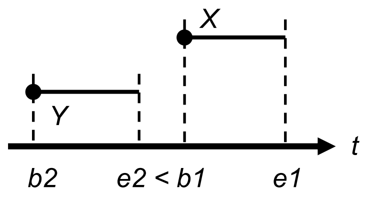
X Contains *Y*



X Meets *Y*



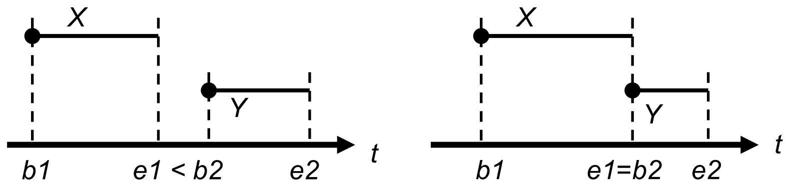




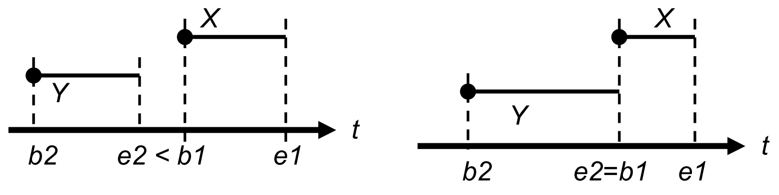
X Precedes *Y*

X Equals *Y*

X Succeeds *Y*



X PrecedesOrMeets *Y*



X SucceedsOrMeets *Y*

Operations to obtain a time interval of a period are as follows:

- year-month interval of X is:
  - (e1 - b1) as a year-month interval.
- day-time interval of X is:
  - (e1 - b1) as a day-time interval.

Set operations on two periods are as follows.

— X Intersect Y returns:

  — [MAX(b1, b2):MIN(e1, e2)) if X Overlaps Y is true.

— X Union Y returns:

  — [MIN(b1, b2):MAX(e1, e2)) if either of X Overlaps Y or X Meets Y is true.

— X Except Y returns:

  — [b1:MIN(b2, e1)) if b1 < b2 and e1 <= e2 are both true.

  — [MAX(e2, b1):e1) if b2 <= b1 and e2 < e1 are both true.

In this Technical Specification, the following user-defined types and methods are provided to specify time-related queries to the history table.

1) User-defined types:

   a) HS_History type,

   b) <TableTypeIdentifier> type.

2) Methods:

   a) HS_Overlaps

   b) HS_Meets

   c) HS_Precedes

   d) HS_PrecedesOrMeets

   e) HS_Succeeds

f) HS_SucceedsOrMeets

g) HS_Contains

h) HS_Equals

i) HS_MonthInterval

j) HS_DayInterval

k) HS_Intersect

l) HS_Union

m) HS_Except

n) HS_PNormalize

4.1.5 Concept of Period Normalization

Period normalization is introduced to provide an aggregate operation for a number of history rows of a history table.

As an example, consider the case where a history table has two tracked columns in addition to the primary key, ID. In this case a history row is inserted into the history table whenever at least one of tracked columns is changed.

Now let us consider an application that requests a history table on only one of the two tracked columns that were originally specified. Since it was possible that tracked rows were added to the originally specified history table for changes in not only the specified (tracked) column but also the other tracked column, the application would have to specify a complicated query to determine the actual periods when the values of the requested column were not changed.

NOTE Some example queries of period normalization are shown in Annex A.3.

Example 1:

History Table for Tracked Table TT1

HS_SEQ	ID	Column_A	Column_B	HS_Hist (HS_BeginTime, HS_EndTime)
1	1	A1	B1	(T1, T2)
2	1	A1	B2	(T2, T3)
3	1	A2	B2	(T3, T4)
4	1	A2	B3	(T4, T5)
5	1	A3	B3	(T5, T6)
6	1	A4	B3	(T6, T7)
7	1	A4	B4	(T7, NULL)

When period normalization of Column_B is executed, as the value of the Column_B is B1 from T1 to T2, B2 from T2 to T4, B3 from T4 to T7, and B4 from T7 to now, the period-normalized table is as follows:

HS_SEQ	ID	Column_B	HS_Hist (HS_BeginTime, HS_EndTime)
1	1	B1	(T1, T2)
2	1	B2	(T2, T4)

4	1	B3	(T4, T7)
7	1	B4	(T7, NULL)

Example 2:

History Table for Tracked Table TT2

HS_SEQ	ID	Column_A	Column_B	Column_C	Column_D	HS_Hist (HS_BeginTime, HS_EndTime)
1	1	A1	B1	C1	D1	(T1, T3)
2	2	A1	B1	C1	D1	(T2, T4)
3	1	A2	B1	C1	D1	(T3, T5)
4	2	A2	B1	C1	D1	(T4, T6)
5	1	A2	B2	C1	D1	(T5, T7)
6	2	A2	B2	C1	D1	(T6, T8)
7	1	A2	B2	C2	D2	(T7, T9)
8	2	A2	B2	C2	D2	(T8, T10)
9	1	A3	B2	C2	D3	(T9, T11)
10	2	A3	B2	C2	D3	(T10, T11)
11	1	A3	B3	C3	D3	(T11, T12)
12	2	A3	B3	C3	D3	(T11, T13)
13	1	A3	B4	C4	D3	(T12, NULL)
14	2	A3	B4	C4	D3	(T13, NULL)

When period normalization of Column_B and Column_C is executed, for the rows whose ID is 1, the values of Column_B and Column_C are B1 and C1 from T1 to T5, B2 and C1 from T5 to T7, B2 and C2 from T7 to T11, B3 and C3 from T11 to T12, and B4 and C4 from T12 to now, and for the rows whose ID is 2, the values of Column_B and Column_C are B1 and C1 from T2 to T6, B2 and C1 from T6 to T8, B2 and C2 from T8 to T11, B3 and C3 from T11 to T13, and B4 and C4 from T13 to now. The period-normalized table is as follows:

HS_SEQ	ID	Column_B	Column_C	HS_Hist (HS_BeginTime, HS_EndTime)
1	1	B1	C1	(T1, T5)
5	1	B2	C1	(T5, T7)
7	1	B2	C2	(T7, T11)
11	1	B3	C3	(T11, T12)
13	1	B4	C4	(T12, NULL)
2	2	B1	C1	(T2, T6)
6	2	B2	C1	(T6, T8)
8	2	B2	C2	(T8, T11)
12	2	B3	C3	(T11, T13)
14	2	B4	C4	(T13, NULL)

The operation of “period normalization” is introduced in this Technical Specification to perform the derivation of a tracked table on a subset of the original set of tracked columns of a history table.

Period normalization of selected columns of a history table is an operation that derives a single history row from one or more history rows.

In order to execute period normalization of selected columns of a history table, the history rows of a history row set are divided into sets of history rows, whose values of the selected columns are not distinct across a set of history rows which have contiguous periods.

In each such set of history rows, from the begin time of the oldest history row in the set to the end time of the latest history row in the set, the values of the selected columns are not distinct. Therefore, the period-normalized history table holds one row, which has the following values, for each set of history rows in each history row set:

1) The values of the selected columns which are common to all of the rows in the set of history rows.

2) The value of the begin time of the oldest history row in the set of history rows.

3) The value of the end time of the latest history row in the set of history rows.

4) Null values for the other tracked columns in the history row.

4.2 Structure of History Table

Whenever an insert/update operation occurs on a row of a tracked table, a row is inserted into a history table. A row of a tracked table is called a tracked row and a row inserted into a history table is called a history row. Such history rows can be grouped as rows corresponding to a certain tracked row. A multiset of history rows corresponding to a certain tracked row is called a history row set of the tracked row.

The following type is provided to represent a period associated with a history row:

— The HS_History type with the following attributes:

— HS_BeginTime: a TIMESTAMP(<*TimestampPrecision*>) value to store a begin time;

— HS_EndTime: a TIMESTAMP(<*TimestampPrecision*>) value to store an end time;

The following conditions are held between the members of a history row set of a tracked row:

— The time order of the values of the begin times for all history rows contained in a history row set is the same as the order that the history rows were added to the history table.

— The begin time of history row A in a history row set is equal to the end time of the history row B in the history row set that was added immediately before history row A was added.

A history row is represented by the following type which is automatically generated for every tracked table:

— The <TableTypeIdentifier> type with the following attributes:

— an attribute HS_SEQ of type <*TypeOf_HS_SEQ*> to contain the sequence number which is associated with a history row;

— one attribute for each tracked column of the tracked table;

— an attribute HS_Hist of type HS_History to contain the period which is associated with a history row.

4.3 Creating History Table and Storing History Row in History Table

The HS_CreateHistory procedure is provided to create a history table corresponding to the specified tracked table.

In order to keep a history table up to date with changes made to the tracked table, the following triggers are created:

— An AFTER trigger to insert a new history row into the history table when a row is inserted into the tracked table.

— An AFTER trigger to update the end time of the latest history row to the transaction timestamp value and insert a new history row into the history table when the value of a tracked column is updated.

— An AFTER trigger to update the end time of the latest history row to the transaction timestamp value when a row is deleted from the tracked table.

These triggers are automatically created for every tracked table when the HS_CreateHistory procedure is invoked.

4.4 Retrieving a History Table

To retrieve a history table, the HS_HistoryTable method or the HS_PNormalize method of <TableTypeIdentifier> type should be used. The HS_HistoryTable method returns the whole history table. The HS_PNormalize method returns the table that is the result of period normalization of specified column(s) of the history table.

4.5 Types representing history rows

4.5.1 HS_History type

4.5.1.1 Attributes of the HS_History type

The HS_History type is a representation of a period, using the following attributes:

— HS_BeginTime: the start of a period;

— HS_EndTime: the end of a period;

A value *V* of the HS_History type is as follows, where *T* is a transaction timestamp that corresponds to the SQL-transaction which the HS_History constructor method is invoked:

a) *V*.HS_BeginTime cannot be null.

b) *V*.HS_BeginTime is not greater than *T*.

c) If *V*.HS_EndTime is null, then *V* represents a period extending from *V*.HS_BeginTime (inclusive) until the present moment.

d) If *V*.HS_EndTime is not null, then *V*.HS_EndTime is not less than *V*.HS_BeginTime but less than or equal to *T*. In this case *V* represents a period extending from *V*.HS_BeginTime (inclusive) through *V*.HS_EndTime (exclusive).

4.5.1.2 Methods of the HS_History type

The type HS_History provides the following methods for public use:

4.5.1.2.1 Constructor method

— HS_History
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
 endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS HS_History:
Generate HS_History value which has the specified TIMESTAMP values as the begin time and the end time.

4.5.1.2.2 Methods for treating a period

— HS_Overlaps
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
 endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value overlaps with the period specified by the arguments;

— HS_Overlaps(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value overlaps with the specified period;

— HS_Meets
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
 endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value meets the period specified by the arguments;

— HS_Meets(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value meets the specified period;

— HS_Precedes
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
 endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value precedes the period specified by the arguments;

— HS_Precedes(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value precedes the specified period;

— HS_PrecedesOrMeets
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
 endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value precedes or meets the period specified by the arguments;

— HS_PrecedesOrMeets(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value precedes or meets the specified period;

— HS_Succeeds
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
 endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value succeeds the period specified by the arguments;

— HS_Succeeds(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value succeeds the specified period;

— HS_SucceedsOrMeets
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value succeeds or meets the period specified by the arguments;

— HS_SucceedsOrMeets(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value succeeds or meets the specified period;

— HS_Contains(timePoint TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value contains the specified TIMESTAMP value;

— HS_Contains
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value contains the period specified by the arguments;

— HS_Contains(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value contains the specified period;

— HS_Equals
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS INTEGER:
Test whether the period of an HS_History value is equal to the period specified by the arguments;

— HS_Equals(hs_hist HS_History)
RETURNS INTEGER:
Test whether the period of an HS_History value is equal to the specified period;

— HS_MonthInterval()
RETURNS INTERVAL YEAR TO MONTH:
Obtain the length of the period of an HS_History value as a year-month interval;

— HS_DayInterval()
RETURNS INTERVAL DAY TO SECOND:
Obtain the length of the period of an HS_History value as a day-time interval;

— HS_Intersect
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS HS_History:
Generate a new HS_History value with the period which is the overlap of the period of an HS_History value and the period specified by the arguments;

— HS_Intersect(hs_hist HS_History)
RETURNS HS_History:
Generate a new HS_History value with the period which is the overlap of the period of an HS_History value and the specified period;

— HS_Union
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS HS_History:
Generate a new HS_History value with the period which is the union of the period of an HS_History value and the period specified by the arguments;

— HS_Union(hs_hist HS_History)
RETURNS HS_History:
Generate a new HS_History value with the period which is the union of the period of an HS_History value and the specified period;

— HS_Except
(beginOfPeriod TIMESTAMP(<*TimestampPrecision*>),
endOfPeriod TIMESTAMP(<*TimestampPrecision*>))
RETURNS HS_History:
Generate a new HS_History value with the period obtained from the period of an HS_History value except for the period specified by the arguments;

— HS_Except(hs_hist HS_History)
RETURNS HS_History:
Generate a new HS_History value with the period obtained from the period of an HS_History value except for the specified period;

4.5.2 <TableTypeIdentifier> type

4.5.2.1 Attributes of the <TableTypeIdentifier> type

The <TableTypeIdentifier> type is an abstraction for attributes of a history row, using the following attributes:

— an attribute HS_SEQ of type <*TypeOf_HS_SEQ*>;

— one attribute for each of the tracked column of the tracked table;

— an attribute HS_Hist of type HS_History;

4.5.2.2 Methods of the <TableTypeIdentifier> type

The type <*TableTypeIdentifier*> provides the following methods:

— HS_HistoryTable()
RETURNS TABLE(HS_SEQ <*TypeOf_HS_SEQ*>, <*AllTrackedColumnsDef*>, HS_Hist HS_History):
Obtain whole history table;

— HS_PNormalize(targetColumn CHARACTER VARYING(<*ColumnNameLength*>))
RETURNS TABLE(HS_SEQ <*TypeOf_HS_SEQ*>, <*AllTrackedColumnsDef*>, HS_Hist HS_History):
Obtain a period-normalized table of the specified one column of a history table;

— HS_PNormalize(targetColumns CHARACTER VARYING(<*ColumnNameLength*>) ARRAY)
RETURNS TABLE(HS_SEQ <*TypeOf_HS_SEQ*>, <*AllTrackedColumnsDef*>, HS_Hist HS_History):
Obtain a period-normalized table of the specified one or more columns of a history table;

4.6 Complementary SQL-invoked regular functions

To ease conformance for implementation of this Technical Specification, each method intended for public use is complemented by an SQL-invoked regular function.

For each such method, the type of specified method, the method name, parameter types (if any), and the name of the corresponding SQL-invoked regular function are listed in Table 1, Table 2, and Table 3 in the following section.

4.6.1 Constructor method of the HS_History type

Table 1 — Method and function name correspondences

Type Name	Method Name	Parameter Types (if any)	Function Name
HS_History	HS_History	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_History

4.6.2 Methods of the HS_History type for treating a period

Table 2 — Method and function name correspondences

Type Name	Method Name	Parameter Types (if any)	Function Name
HS_History	HS_Overlaps	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Overlaps_TT
HS_History	HS_Overlaps	HS_History	HS_Overlaps_H
HS_History	HS_Meets	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Meets_TT
HS_History	HS_Meets	HS_History	HS_Meets_H
HS_History	HS_Precedes	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Precedes_TT
HS_History	HS_Precedes	HS_History	HS_Precedes_H
HS_History	HS_PrecedesOrMeets	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_PrecedesOrMeets_TT
HS_History	HS_PrecedesOrMeets	HS_History	HS_PrecedesOrMeets_H
HS_History	HS_Succeeds	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Succeeds_TT
HS_History	HS_Succeeds	HS_History	HS_Succeeds_H
HS_History	HS_SucceedsOrMeets	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_SucceedsOrMeets_TT
HS_History	HS_SucceedsOrMeets	HS_History	HS_SucceedsOrMeets_H
HS_History	HS_Contains	TIMESTAMP(<*TimestampPrecision*>)	HS_Contains_T
HS_History	HS_Contains	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Contains_TT
HS_History	HS_Contains	HS_History	HS_Contains_H
HS_History	HS_Equals	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Equals_TT
HS_History	HS_Equals	HS_History	HS_Equals_H
HS_History	HS_MonthInterval		HS_MonthInterval
HS_History	HS_DayInterval		HS_DayInterval
HS_History	HS_Intersect	TIMESTAMP(<*TimestampPrecision*>),	HS_Intersect_TT

Type Name	Method Name	Parameter Types (if any)	Function Name
		TIMESTAMP(<*TimestampPrecision*>)	
HS_History	HS_Intersect	HS_History	HS_Intersect_H
HS_History	HS_Union	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Union_TT
HS_History	HS_Union	HS_History	HS_Union_H
HS_History	HS_Except	TIMESTAMP(<*TimestampPrecision*>), TIMESTAMP(<*TimestampPrecision*>)	HS_Except_TT
HS_History	HS_Except	HS_History	HS_Except_H

4.6.3 Methods of the <TableTypeIdentifier> type

Table 3 — Method and function name correspondences

Type Name	Method Name	Parameter Types (if any)	Function Name
<*TableTypeIdentifier*>	HS_HistoryTable		HS_HistoryTable
<*TableTypeIdentifier*>	HS_PNormalize	CHARACTER VARYING(<*ColumnNameLength*>)	HS_PNormalize
<*TableTypeIdentifier*>	HS_PNormalize	CHARACTER VARYING(<*ColumnNameLength*>) ARRAY	HS_PNormalize_A

4.7 The History Information Schema

This Technical Specification prescribes an Information Schema called HS_INFORMTN_SCHEMA. It contains views for the following purposes:

— a view HS_TRACKED_TABLES, which lists the tables that have an associated history table;

— a view HS_TRACKED_COLUMNS, which lists the columns that are specified as the parameter of HS_CreateHistory procedure.

5 History Procedures

5.1 HS_CreateHistory Procedure and its related Procedures

5.1.1 HS_CreateHistory Procedure

Purpose

Create a history table for the specified tracked table, create three triggers to maintain the history table, define the type corresponding to the history table, and initialize the history table.

Definition

```
CREATE PROCEDURE HS_CreateHistory
    (IN TableName CHARACTER VARYING(<TableNameLength>),
     IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
```

```
      LANGUAGE SQL
      NOT DETERMINISTIC
      MODIFIES SQL DATA
      SQL SECURITY INVOKER
      BEGIN
         CALL HS_CreateHistoryErrorCheck(TableName, TrackedColumns);
         CALL HS_CreateHistoryPrivilegeCheck(TableName, TrackedColumns);
         CALL HS_CreateHistoryTableSequenceNumberGenerator(TableName);
         CALL HS_CreateHistoryTableType(TableName, TrackedColumns);
         CALL HS_CreateHistoryTable(TableName, TrackedColumns);
         CALL HS_CreateInsertTrigger(TableName, TrackedColumns);
         CALL HS_CreateUpdateTrigger(TableName, TrackedColumns);
         CALL HS_CreateDeleteTrigger(TableName, TrackedColumns);
         CALL HS_CreateHistoryTableMethod(TableName, TrackedColumns);
         CALL HS_CreatePNormalizeMethod(TableName, TrackedColumns);
         CALL HS_InitializeHistoryTable(TableName, TrackedColumns);
      END
```

Definitional Rules

1) The SQL-session context shall be such that:

   a) The value of the current default catalog name is <*CatalogName*>.

   b) The value of the current default unqualified schema name is <*SchemaName*>.

2) The enabled authorization identifiers shall include the authorization identifier of schema whose name is <*CatalogName*> . <*SchemaName*>.

NOTE An enabled authorization identifier is defined in ISO/IEC 9075.

3) If an error occurred in one of procedures invoked in the HS_CreateHistory procedure, the HS_CreateHistory procedure is implicitly ended with a <rollback statement>.

Description

1) The *HS_CreateHistory(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* takes the following input parameters:

   a) *TableName*, whose value is a character representation of a table name which conforms to the Format and Syntax Rules of <table name> in ISO/IEC 9075-2.

   b) *TrackedColumns*, each element of whose value is a character representation of a column name which conforms to the Format and Syntax Rules of <column name> in ISO/IEC 9075-2.

5.1.2 HS_CreateHistoryErrorCheck Procedure

Purpose

Check whether the input parameters of the HS_CreateHistory procedure are valid.

Definition

```
   CREATE PROCEDURE HS_CreateHistoryErrorCheck
      (IN TableName CHARACTER VARYING(<TableNameLength>),
       IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
      LANGUAGE SQL
```

```
DETERMINISTIC
READS SQL DATA
BEGIN
   DECLARE IdentifierColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY;
   DECLARE i INTEGER;
   DECLARE j INTEGER;

   --
   -- Check if the specified TableName parameter is valid
   --
   IF TableName IS NULL THEN
      SIGNAL SQLSTATE '2FF01' SET MESSAGE_TEXT =
         'table name is a null value';
   END IF;
   --
   -- Check if the table specified exists in the SQL-database
   --
   IF NOT EXISTS(SELECT *
         FROM information_schema.tables
         WHERE TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
            AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
            AND TABLE_NAME = HS_ExtractTableIdentifier(TableName))
         THEN
      SIGNAL SQLSTATE '2FF02' SET MESSAGE_TEXT =
         'table does not exist';
   END IF;
   --
   -- Check if the specified table has
   --  unique constraints with NOT NULL for all columns of the constraint.
   -- If there is no unique constraint with NOT NULL,
   --  then exception condition is raised.
   --
   CALL HS_GetHistoryRowSetIdentifierColumns(
         TableName, TrackedColumns, IdentifierColumns);
   IF NOT EXISTS(SELECT *
         FROM UNNEST(IdentifierColumns) AS CLMS(CNAME)) THEN
      SIGNAL SQLSTATE '2FF13' SET MESSAGE_TEXT =
         'tracked table has no unique constraint with NOT NULL';
   END IF;
   --
   -- Check if the history table for the specified table does not exist
   --
   IF EXISTS(SELECT *
         FROM information_schema.tables
         WHERE TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
            AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
            AND TABLE_NAME = HS_ConstructTableIdentifier(TableName))
         THEN
      SIGNAL SQLSTATE '2FF03' SET MESSAGE_TEXT =
         'history table already exists';
   END IF;
   --
```

```
      -- Check if the specified TrackedColumns parameter is valid
      --
      IF TrackedColumns IS NULL THEN
         SIGNAL SQLSTATE '2FF04' SET MESSAGE_TEXT =
            'no tracked column is specified';
      END IF;
      IF CARDINALITY(TrackedColumns) < 1 THEN
         SIGNAL SQLSTATE '2FF04' SET MESSAGE_TEXT =
            'no tracked column is specified';
      END IF;
      --
      -- For each element of an array TrackedColumns, check if
      --   a) it is not a null value.
      --   b) it is a column name of the specified table.
      --
      SET i = 1;
      WHILE i <= CARDINALITY(TrackedColumns) DO
         IF TrackedColumns[i] IS NULL THEN
         -- Specified column name is null.
            SIGNAL SQLSTATE '2FF05' SET MESSAGE_TEXT =
               'column name is a null value';
         ELSEIF NOT EXISTS(SELECT *
               FROM information_schema.columns
               WHERE
                  TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
                  AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
                  AND TABLE_NAME = HS_ExtractTableIdentifier(TableName)
                  AND column_name =
                     HS_ExtractColumnIdentifier(TrackedColumns[i])) THEN
         -- The tracked table does not have a specified column.
            SIGNAL SQLSTATE '2FF06' SET MESSAGE_TEXT =
               'column does not exist';
         END IF;
         SET i = i + 1;
      END WHILE;
      --
      -- Check if all unique constraint columns are specified
      -- as the TrackedColumns parameter
      --
      SET i = 1;
      WHILE i <= CARDINALITY(IdentifierColumns) DO
         IF NOT EXISTS(SELECT *
                  FROM UNNEST(TrackedColumns) AS CLMS(CNAME)
                  WHERE CNAME = IdentifierColumns[i]) THEN
         -- i-th identifier column is not specified as tracked columns.
            SIGNAL SQLSTATE '2FF14' SET MESSAGE_TEXT =
               'some of unique constraint columns are not specified as tracked
               columns';
         END IF;
         SET i = i + 1;
      END WHILE;
   END
```

Description

1) The *HS_CreateHistoryErrorCheck(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.3 HS_CreateHistoryPrivilegeCheck procedure

Purpose

Check whether or not the definer has required privileges on columns composing the primary key of the tracked table and tracked columns.

Definition

```
CREATE PROCEDURE HS_CreateHistoryErrorCheck
   (IN TableName CHARACTER VARYING(<TableNameLength>),
    IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   DETERMINISTIC
   READS SQL DATA
   SQL SECURITY INVOKER
   BEGIN
      DECLARE M_TrackedColumns CHARACTER VARYING(<ColumnNameLength>) MULTISET;

      DECLARE i INTEGER;

      --
      -- Construct a multiset consisting of names of tracked columns
      --
      SET i = 1;
      WHILE i <= CARDINALITY(TrackedColumns) DO
         SET M_TrackedColumns[i] =
         HS_ExtractColumnIdentifier(TrackedColumn[i]);
         SET i = i + 1;
      END WHILE;
      --
      -- Check the condition that the multiset of names of
      --   columns composing tracked columns
      --   is contained that of names of columns
      --   on which the definer of a history has SELECT privilege
      --
      IF M_TrackedColumns NOT SUBMULTISET OF
            MULTISET
            (SELECT COLUMN_NAME
               FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
               WHERE TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
                  AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
                  AND TABLE_NAME = HS_ExtractTableIdentifier(TableName))
            THEN
         -- Raise an exception condition
         SIGNAL SQLSTATE '2FF26' SET MESSAGE_TEXT =
            'lack of privileges';
      END IF;
   END
```

Description

1) The *HS_CreateHistoryPrivilegeCheck(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.4 HS_CreateHistoryTableSequenceNumberGenerator procedure

Purpose

Create a sequence generator for generating a sequence number for a history row of the history table.

Definition

```
CREATE PROCEDURE HS_CreateHistoryTableSequenceNumberGenerator
   (IN TableName CHARACTER VARYING(<TableNameLength>))
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpSequenceGeneratorIdentifier
         CHARACTER VARYING(<IdentifierLength>);

      SET tmpSequenceGeneratorIdentifier =
         HS_ConstructSequenceGeneratorIdentifier(TableName);

      SET stmt =
         'CREATE SEQUENCE ' || tmpSequenceGeneratorIdentifier              ||
         ' AS <TypeOf_HS_SEQ> START WITH 1'                                ||
         ' INCREMENT BY 1'                                                 ||
         ' NO MAXVALUE'                                                    ||
         ' NO CYCLE'                                                       ;
      EXECUTE IMMEDIATE stmt;
   END
```

Definitional Rules

1) <TypeOf_HS_SEQ> is the implementation-defined <data type or domain name> of the value generated by the sequence generator.

Description

1) The *HS_CreateHistoryTableSequenceNumberGenerator(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*.

5.1.5 HS_CreateHistoryTableType Procedure

Purpose

Create the <TableTypeIdentifier> type, which is the base type of a history table corresponding to the specified tracked table.

Definition

```
CREATE PROCEDURE HS_CreateHistoryTableType
    (IN TableName CHARACTER VARYING(<TableNameLength>),
     IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpAllTrackedColumnsDef CLOB;

      CALL HS_CreateCommaSeparatedTrackedColumnAndTypeList(
         TableName, TrackedColumns, tmpAllTrackedColumnsDef);

      -- Construct a CREATE TYPE statement
      --    in order to create <TableTypeIdentifier> type
      --    for a basis of the history table of the specified tracked table.
      SET stmt =
         'CREATE TYPE ' || HS_ConstructTableTypeIdentifier(TableName)       ||
         ' AS(HS_SEQ <TypeOf_HS_SEQ>'                                        ||
         ', ' || tmpAllTrackedColumnsDef                                     ||
         '       , HS_Hist HS_History)'                                      ||
         ' STATIC METHOD HS_HistoryTable()'                                  ||
         '  RETURNS TABLE(HS_SEQ <TypeOf_HS_SEQ>'                            ||
         ', ' || tmpAllTrackedColumnsDef                                     ||
         '       , HS_Hist HS_History),'                                     ||
         ' STATIC METHOD HS_PNormalize('                                     ||
         '     targetColumn CHARACTER VARYING(<ColumnNameLength>))'          ||
         '  RETURNS TABLE(HS_SEQ <TypeOf_HS_SEQ>'                            ||
         ', ' || tmpAllTrackedColumnsDef                                     ||
         '       , HS_Hist HS_History),'                                     ||
         ' STATIC METHOD HS_PNormalize('                                     ||
         '     targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)'   ||
         '  RETURNS TABLE(HS_SEQ <TypeOf_HS_SEQ>'                            ||
         ', ' || tmpAllTrackedColumnsDef                                     ||
         '       , HS_Hist HS_History)'                                      ;
      EXECUTE IMMEDIATE stmt;
   END
```

Description

1) The definition of *<TableTypeIdentifier>* type, which is defined in the *HS_CreateHistoryTableType* procedure, is specified in subclause 6.2.1, "<TableTypeIdentifier> Type".

2) The *HS_CreateHistoryTableType(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.6 HS_CreateHistoryTable Procedure

Purpose

Create a history table corresponding to the specified tracked table.

Definition

```
CREATE PROCEDURE HS_CreateHistoryTable
   (IN TableName CHARACTER VARYING(<TableNameLength>),
    IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpAllIdentifierColumns CLOB;
      DECLARE tmpAllIdentifierColumnsOpt CLOB;

      CALL HS_CreateCommaSeparatedIdentifierColumnList(
         TableName, TrackedColumns, NULL, tmpAllIdentifierColumns);
      CALL HS_CreateCommaSeparatedIdentifierColumnList(
         TableName, TrackedColumns, ' WITH OPTIONS NOT NULL',
         tmpAllIdentifierColumnsOpt);

      -- Construct a CREATE TABLE statement
      --   in order to create the history table
      --   according to the specified tracked table.
      SET stmt =
         'CREATE TABLE ' || HS_ConstructTableIdentifier(TableName)          ||
         '  OF ' || HS_ConstructTableTypeIdentifier(TableName)              ||
         '  (REF IS "HS_REF" SYSTEM GENERATED,'                             ||
                tmpAllIdentifierColumnsOpt || ','                           ||
         '    PRIMARY KEY(HS_SEQ)'                                          ||
         '  )'                                                              ;
      EXECUTE IMMEDIATE stmt;
   END
```

Description

1) The *HS_CreateHistoryTable(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.7 HS_CreateInsertTrigger Procedure

Purpose

Create a trigger to insert a history row into the history table when a row is inserted into the corresponding tracked table.

Definition

```
CREATE PROCEDURE HS_CreateInsertTrigger
```

```
(IN TableName CHARACTER VARYING(<TableNameLength>),
 IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
LANGUAGE SQL
NOT DETERMINISTIC
MODIFIES SQL DATA
BEGIN
   DECLARE stmt CLOB;

   DECLARE tmpAllTrackedColumns CLOB;
   DECLARE tmpAllTrackedColumns_nwr CLOB;

   CALL HS_CreateCommaSeparatedTrackedColumnList(
      TrackedColumns, NULL, tmpAllTrackedColumns);
   CALL HS_CreateCommaSeparatedTrackedColumnList(
      TrackedColumns, 'nwr.', tmpAllTrackedColumns_nwr);

   -- Construct a CREATE TRIGGER statement
   --   in order to create an INSERT trigger.
   -- This INSERT trigger inserts a history row into the history table
   --   when a row is inserted into the tracked table.
   SET stmt =
      'CREATE TRIGGER ' || HS_ConstructInsTriggerIdentifier(TableName)    ||
      '  AFTER INSERT ON ' || TableName                                   ||
      '  REFERENCING NEW ROW AS nwr'                                      ||
      '  FOR EACH ROW'                                                    ||
      '  BEGIN ATOMIC'                                                    ||
      '    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);'                ||
      '    DECLARE histVal HS_History;'                                   ||
      '    SET CurTS = HS_GetTransactionTimestamp();'                     ||
      '    SET histVal = NEW HS_History('                                 ||
      '        CurTS, CAST(NULL AS TIMESTAMP(<TimestampPrecision>)));'    ||
      '    INSERT INTO ' || HS_ConstructTableIdentifier(TableName)        ||
      '        (HS_SEQ, ' || tmpAllTrackedColumns || ' , HS_Hist' || ') '||
      '      VALUES ('                                                    ||
      '        (NEXT VALUE FOR '                                          ||
                 HS_ConstructSequenceGeneratorIdentifier(TableName) || '),
      ' ||
               tmpAllTrackedColumns_nwr || ' , histVal);'                 ||
      '  END'                                                             ;
   EXECUTE IMMEDIATE stmt;
END
```

Description

1) The *HS_CreateInsertTrigger(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.8 HS_CreateUpdateTrigger Procedure

Purpose

Create a trigger to insert a history row into the history table when any value of the specified columns of the tracked table is updated.

Definition

```
CREATE PROCEDURE HS_CreateUpdateTrigger
   (IN TableName CHARACTER VARYING(<TableNameLength>),
    IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllIdentifierColumnsComparison_HT1_nwr CLOB;
      DECLARE tmpAllTrackedColumns CLOB;
      DECLARE tmpAllTrackedColumns_odr CLOB;
      DECLARE tmpAllTrackedColumns_nwr CLOB;
      DECLARE tmpAllIdentifierColumns_odr CLOB;
      DECLARE tmpAllIdentifierColumns_nwr CLOB;

      CALL HS_CreateIdentifierColumnSelfJoinCondition(
         TableName, TrackedColumns, 'HT1.', 'nwr.',
         tmpAllIdentifierColumnsComparison_HT1_nwr);

      CALL HS_CreateCommaSeparatedTrackedColumnList(
         TrackedColumns, NULL, tmpAllTrackedColumns);
      CALL HS_CreateCommaSeparatedTrackedColumnList(
         TrackedColumns, 'odr.', tmpAllTrackedColumns_odr);
      CALL HS_CreateCommaSeparatedTrackedColumnList(
         TrackedColumns, 'nwr.', tmpAllTrackedColumns_nwr);
      CALL HS_CreateCommaSeparatedIdentifierColumnList(
         TableName, TrackedColumns, 'odr.', tmpAllIdentifierColumns_odr);
      CALL HS_CreateCommaSeparatedIdentifierColumnList(
         TableName, TrackedColumns, 'nwr.', tmpAllIdentifierColumns_nwr);

      -- Construct a CREATE TRIGGER statement
      --   in order to create an UPDATE trigger.
      -- This UPDATE trigger inserts a history row into the history table
      --   when a row in the tracked table is updated.
      SET stmt =
         'CREATE TRIGGER ' || HS_ConstructUpdTriggerIdentifier(TableName)    ||
         '  AFTER UPDATE OF ' || tmpAllTrackedColumns || ' ON ' || TableName ||
         '  REFERENCING NEW ROW AS nwr'                                      ||
         '              OLD ROW AS odr'                                      ||
         '  FOR EACH ROW'                                                    ||
         '  WHEN((' || tmpAllTrackedColumns_odr || ' ) IS DISTINCT FROM ('   ||
                  tmpAllTrackedColumns_nwr || '))'                           ||
         '  BEGIN ATOMIC'                                                    ||
         '    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);'                ||
         '    DECLARE histVal HS_History; '                                  ||
         '    DECLARE histBegTime TIMESTAMP(<TimestampPrecision>); '         ||
         '    IF ((' || tmpAllIdentifierColumns_odr || ') IS DISTINCT FROM' ||
         '        (' || tmpAllIdentifierColumns_nwr || ')) THEN'             ||
         '      SIGNAL SQLSTATE '2FF27' SET MESSAGE_TEXT ='                  ||
         '        'update operation for identifier columns is not supported';'||
         '    END IF;'                                                       ||
         '    SET CurTS = HS_GetTransactionTimestamp(); '                    ||
         '    SET histVal = NEW HS_History('                                 ||
         '        CurTS, CAST(NULL AS TIMESTAMP(<TimestampPrecision>))); '   ||
         '    UPDATE ' || HS_ConstructTableIdentifier(TableName) || ' HT1'   ||
         '      SET HT1.HS_Hist.HS_EndTime = CurTS'                          ||
         '      WHERE ' || tmpAllIdentifierColumnsComparison_HT1_nwr         ||
         '         AND HT1.HS_Hist.HS_EndTime IS NULL;'                 ||
         '    INSERT INTO ' || HS_ConstructTableIdentifier(TableName)   ||
         '        (HS_SEQ, ' || tmpAllTrackedColumns || ', HS_Hist) '   ||
         '      VALUES ('                                               ||
         '        (NEXT VALUE FOR '                                     ||
                   HS_ConstructSequenceGeneratorIdentifier(TableName) || '),
         ' ||
                 tmpAllTrackedColumns_nwr || ', histVal); '             ||
         '  END'                                                        ;
      EXECUTE IMMEDIATE stmt;
   END
```

Description

1) The *HS_CreateUpdateTrigger(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.9 HS_CreateDeleteTrigger Procedure

Purpose

Create a trigger to update the end time of the latest history row to current transaction timestamp value when a row is deleted from the corresponding tracked table.

Definition

```
CREATE PROCEDURE HS_CreateDeleteTrigger
    (IN TableName CHARACTER VARYING(<TableNameLength>),
     IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpAllIdentifierColumnsComparison_HT1_odr CLOB;

   CALL HS_CreateIdentifierColumnSelfJoinCondition(
         TableName, TrackedColumns, 'HT1.', 'odr.',
         tmpAllIdentifierColumnsComparison_HT1_odr);

      -- Construct a CREATE TRIGGER statement
      --   in order to create a DELETE trigger.
      -- This DELETE trigger updates the end time of the latest history row
      --    according to the tracked row which is deleted from the tracked
      table.
      SET stmt =
         'CREATE TRIGGER ' || HS_ConstructDelTriggerIdentifier(TableName)   ||
         '  AFTER DELETE ON ' || TableName                                  ||
         '  REFERENCING OLD ROW AS odr'                                     ||
         '  FOR EACH ROW'                                                   ||
         '  BEGIN ATOMIC'                                                   ||
         '    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);'               ||
         '    SET CurTS = HS_GetTransactionTimestamp();'                    ||
         '    UPDATE ' || HS_ConstructTableIdentifier(TableName) || ' HT1'  ||
         '      SET HT1.HS_Hist.HS_EndTime = CurTS'                         ||
         '      WHERE ' || tmpAllIdentifierColumnsComparison_HT1_odr        ||
         '          AND HT1.HS_Hist.HS_EndTime IS NULL;'                    ||
         '  END'                                                             ;
      EXECUTE IMMEDIATE stmt;
   END
```

Description

1) The *HS_CreateDeleteTrigger(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.10 HS_CreateHistoryTableMethod Procedure

Purpose

Create a method to obtain a history table.

Definition

```
   CREATE PROCEDURE HS_CreateHistoryTableMethod
      (IN TableName CHARACTER VARYING(<TableNameLength>),
       IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
      LANGUAGE SQL
      DETERMINISTIC
      MODIFIES SQL DATA
      BEGIN
         DECLARE stmt CLOB;
         DECLARE tmpAllTrackedColumns CLOB;
         DECLARE tmpAllTrackedColumnsDef CLOB;
         DECLARE tmpAllTrackedColumnLiterals CLOB;

         CALL HS_CreateCommaSeparatedTrackedColumnList(
            TrackedColumns, NULL, tmpAllTrackedColumns);
         CALL HS_CreateCommaSeparatedTrackedColumnAndTypeList(
            TableName, TrackedColumns, tmpAllTrackedColumnsDef);
         CALL HS_CreateCommaSeparatedTrackedColumnLiteralList(
            TableName, TrackedColumnLiterals);

         -- Construct a CREATE STATIC METHOD statement
         --    in order to create the HS_HistoryTable() method
         --    for <TableTypeIdentifier> type.
         SET stmt =
            'CREATE STATIC METHOD HS_HistoryTable()'                        ||
            '  RETURNS TABLE('                                              ||
            '      HS_SEQ <TypeOf_HS_SEQ>, '                                ||
                   tmpAllTrackedColumnsDef || ', HS_Hist HS_History)'       ||
            '  FOR ' || HS_ConstructTableTypeIdentifier(TableName)          ||
            '  BEGIN'                                                       ||
            '    IF MULTISET[' || tmpAllTrackedColumnLiterals || ']'        ||
            '        NOT SUBMULTISET OF MULTISET' || '('                    ||
            '          SELECT COLUMN_NAME'                                  ||
            '            FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES'         ||
      '            WHERE TABLE_CATALOG = '                                 ||
                           HS_ConstructCatalogIdentifierLiteral(TableName)||
      '                AND TABLE_SCHEMA = '                                ||
                           HS_ConstructSchemaIdentifierLiteral(TableName)||
      '                AND TABLE_NAME = '                                  ||
                           HS_ConstructTableIdentifierLiteral(TableName)  ||
      '         ) '                                                        ||
      '     THEN'                                                          ||
      '       SIGNAL SQLSTATE ''2FF26'' SET MESSAGE_TEXT = '               ||
      '         ''lack of privileges''; '                                  ||
      '     END IF;'                                                       ||
      '     RETURN TABLE('                                                 ||
      '       SELECT HS_SEQ, ' || tmpAllTrackedColumns || ', HS_Hist'      ||
      '         FROM ' || HS_ConstructTableIdentifier(TableName) || ') AS
      T;' ||
      '   END'                                                             ;
   EXECUTE IMMEDIATE stmt;
END
```

Description

1) The definition of *HS_HistoryTable* method of <*TableTypeIdentifier*> type, which is defined in the *HS_CreateHistoryTableMethod* procedure, is specified in subclause 6.2.2, "HS_HistoryTable Method".

2) The *HS_CreateHistoryTableMethod(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.11 HS_CreatePNormalizeMethod Procedure

Purpose

Create methods to obtain a period-normalized table of the specified columns for a history table.

Definition

```
CREATE PROCEDURE HS_CreatePNormalizeMethod
    (IN TableName CHARACTER VARYING(<TableNameLength>),
     IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpAllTrackedColumns CLOB;
      DECLARE tmpAllTrackedColumns_HT1 CLOB;
      DECLARE tmpAllTrackedColumnsDef CLOB;
      DECLARE tmpAllTrackedColumnLiterals CLOB;
      DECLARE tmpAllIdentifierColumns CLOB;
      DECLARE tmpAllIdentifierColumnsComparison_HT1_HT2 CLOB;
      DECLARE tmpTableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);
      DECLARE   AllIdentifierColumns   CHARACTER   VARYING(<ColumnNameLength>)
      ARRAY;
      DECLARE tmpCatalogIdentifierLiteral CLOB;
      DECLARE tmpSchemaIdentifierLiteral CLOB;
DECLARE tmpTableIdentifierLiteral CLOB;

SET tmpTableTypeIdentifier = HS_ConstructTableTypeIdentifier(TableName);
SET tmpCatalogIdentifierLiteral =
   HS_ConstructCatalogIdentifierLiteral(TableName);
SET tmpSchemaIdentifierLiteral =
   HS_ConstructSchemaIdentifierLiteral(TableName);
SET tmpTableIdentifierLiteral =
   HS_ConstructTableIdentifierLiteral(TableName);
CALL HS_CreateCommaSeparatedTrackedColumnList(
   TrackedColumns, NULL, tmpAllTrackedColumns);
CALL HS_CreateCommaSeparatedTrackedColumnList(
   TrackedColumns, 'HT1.', tmpAllTrackedColumns_HT1);
CALL HS_CreateCommaSeparatedTrackedColumnAndTypeList(
   TableName, TrackedColumns, tmpAllTrackedColumnsDef);
CALL HS_CreateCommaSeparatedTrackedColumnLiteralList(
   TableName, TrackedColumnLiterals);
CALL HS_CreateCommaSeparatedIdentifierColumnList(
   TableName, TrackedColumns, NULL,
   tmpAllIdentifierColumns);
CALL HS_CreateIdentifierColumnSelfJoinCondition(
   TableName, TrackedColumns, 'HT1.', 'HT2.',
   tmpAllIdentifierColumnsComparison_HT1_HT2);

CALL HS_GetHistoryRowSetIdentifierColumns(
   TableName, TrackedColumns, AllIdentifierColumns);
-- In case only one tracked column except unique constraint columns,
IF  CARDINALITY(TrackedColumns)  -  CARDINALITY(AllIdentifierColumns)  =  1
THEN
   -- Create HS_Pnormalize method with a non-ARRAY parameter.
   SET stmt =
   !! The value set into SQL variable "stmt" is
   !! the character string value, which content is
   !! the 1st CREATE METHOD statement in the DEFINITION of
   !! Subclause 6.2.3, "HS_PNormalize Methods",
   !! with the value of SQL variable "tmpAllTrackedColumnsDef"
   !!        as instance of <AllTrackedColumnsDef> ,
   !!      the value of SQL variable "tmpTableTypeIdentifier"
   !!        as instance of <TableTypeIdentifier>.
   EXECUTE IMMEDIATE stmt;

   -- Create HS_Pnormalize method with an ARRAY parameter.
   SET stmt =
   !! The value set into SQL variable "stmt" is
   !! the character string value, which content is
   !! the 2nd CREATE METHOD statement in the DEFINITION of
   !! Subclause 6.2.3, "HS_PNormalize Methods",
   !! with the value of SQL variable "tmpAllTrackedColumnsDef"
```

```
      !!        as instance of <AllTrackedColumnsDef> ,
      !!      the value of SQL variable "tmpTableTypeIdentifier"
      !!        as instance of <TableTypeIdentifier>.
      EXECUTE IMMEDIATE stmt;
   -- In case more than one tracked columns except unique constraint
   columns,
   ELSE
      -- Create HS_Pnormalize method with a non-ARRAY parameter.
      SET stmt =
      !! The value set into SQL variable "stmt" is
      !! the character string value, which content is
      !! the 1st CREATE METHOD statement in the DEFINITION of
      !! Subclause 6.2.3, "HS_PNormalize Methods",
      !! with the value of SQL variable "tmpAllTrackedColumnsDef"
      !!        as instance of <AllTrackedColumnsDef> ,
      !!      the value of SQL variable "tmpTableTypeIdentifier"
      !!        as instance of <TableTypeIdentifier>.
      EXECUTE IMMEDIATE stmt;

      -- Create HS_Pnormalize method with an ARRAY parameter.
      SET stmt =
      !! The value set into SQL variable "stmt" is
      !! the character string value, which content is
      !! the 3rd CREATE METHOD statement in the DEFINITION of
      !! Subclause 6.2.3, "HS_PNormalize Methods",
      !! with the value of SQL variable "tmpAllTrackedColumnsDef"
      !!        as instance of <AllTrackedColumnsDef> ,
      !!      the value of SQL variable "tmpTableTypeIdentifier"
      !!        as instance of <TableTypeIdentifier>,
      !!      the value of SQL variable "tmpAllTrackedColumnLiterals"
      !!        as instance of <TrackedColumnLiterals>,
      !!      the value of SQL variable "tmpCatalogIdentifierLiteral"
      !!        as instance of <CatalogIdentifierLiteral>,
      !!      the value of SQL variable "tmpSchemaIdentifierLiteral"
      !!        as instance of <SchemaIdentifierLiteral>,
      !!      the value of SQL variable "tmpTableIdentifierLiteral"
      !!        as instance of <TableIdentifierLiteral>,
      !!      the value of SQL variable "tmpAllTrackedColumns"
      !!        as instance of <AllTrackedColumns>,
      !!      the value of SQL variable "tmpAllTrackedColumns_HT1"
      !!        as instance of <AllTrackedColumns_HT1>,
      !!      the value of SQL variable "tmpAllIdentifierColumns"
      !!        as instance of <AllIdentifierColumns>,
      !!      the value of SQL variable
      !!                    "tmpAllIdentifierColumnsComparison_HT1_HT2"
      !!        as instance of <AllIdentifierColumnsComparison_HT1_HT2>.
      EXECUTE IMMEDIATE stmt;
   END IF;
END
```

Description

1) The definition of *HS_PNormalize* method of <*TableTypeIdentifier*> type, which is defined in the *HS_CreatePNormalizeMethod* procedure, is specified in subclause 6.2.3, "HS_PNormalize Methods".

2) The *HS_CreatePNormalizeMethod(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

5.1.12 HS_InitializeHistoryTable Procedure

Purpose

Initialize the history table.

Definition

```
CREATE PROCEDURE HS_InitializeHistoryTable
   (IN TableName CHARACTER VARYING(<TableNameLength>),
    IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpAllTrackedColumns CLOB;

      CALL HS_CreateCommaSeparatedTrackedColumnList(
         TrackedColumns, NULL, tmpAllTrackedColumns);

      -- Construct an INSERT statement
      --   in order to insert all rows in the tracked table
      --   into the history table.
      SET stmt =
         'INSERT INTO ' || HS_ConstructTableIdentifier(TableName)           ||
         '    (HS_SEQ, ' || tmpAllTrackedColumns || ', HS_Hist)'            ||
         '  SELECT (NEXT VALUE FOR ' ||
                  HS_ConstructSequenceGeneratorIdentifier(TableName) || ')'
         ||
         '        , ' || tmpAllTrackedColumns                                ||
         '        , NEW HS_History('                                         ||
         '              HS_GetTransactionTimestamp()'                        ||
         '            , CAST(NULL AS TIMESTAMP(<TimestampPrecision>)))'      ||
         '    FROM ' || TableName                                            ;
      EXECUTE IMMEDIATE stmt;
   END
```

Description

1) The *HS_InitializeHistoryTable(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) value *TrackedColumns*.

5.2 HS_DropHistory Procedure and its related Procedures

5.2.1 HS_DropHistory Procedure

Purpose

Drop the history table and the history table type for the specified tracked table. Further, drop the triggers and the methods that refer to that history table.

Definition

```
CREATE PROCEDURE HS_DropHistory
   (IN TableName CHARACTER VARYING(<TableNameLength>))
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      CALL HS_DropHistoryErrorCheck(TableName);
      CALL HS_DropHistoryTableTypeMethod(TableName);
      CALL HS_DropHistoryTrigger(TableName);
      CALL HS_DropHistoryTable(TableName);
      CALL HS_DropHistoryTableType(TableName);
      CALL HS_DropHistoryTableSequenceNumberGenerator(TableName);
   END
```

Description

1) The *HS_DropHistory(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) *TableName*, whose value is a character representation of a table name which confirms to the Format and Syntax Rules of <table name> in ISO/IEC 9075-2.

5.2.2 HS_DropHistoryErrorCheck Procedure

Purpose

Check whether the input parameters of the HS_DropHistory procedure are valid.

Definition

```
CREATE PROCEDURE HS_DropHistoryErrorCheck
   (IN TableName CHARACTER VARYING(<TableNameLength>))
   LANGUAGE SQL
   DETERMINISTIC
   READS SQL DATA
   BEGIN
      -- Check if or not null value is specified as a table name.
      IF TableName IS NULL THEN
         SIGNAL SQLSTATE '2FF01' SET MESSAGE_TEXT =
            'table name is a null value';
      END IF;
      -- Check if or not specified tracked table does not exist.
      IF NOT EXISTS(SELECT *
         FROM information_schema.tables
         WHERE TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
```

```
            AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
            AND TABLE_NAME = HS_ExtractTableIdentifier(TableName)) THEN
         SIGNAL SQLSTATE '2FF02' SET MESSAGE_TEXT =
            'table does not exist';
      END IF;
      -- Check if or not the history table according to the tracked table
      --   does not exist.
      IF NOT EXISTS(SELECT *
         FROM information_schema.tables
         WHERE TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
            AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
            AND TABLE_NAME = HS_ConstructTableIdentifier(TableName)) THEN
         SIGNAL SQLSTATE '2FF07' SET MESSAGE_TEXT =
            'history table does not exist';
      END IF;
   END
```

Description

1) The *HS_DropHistoryErrorCheck(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

5.2.3 HS_DropHistoryTableTypeMethod Procedure

Purpose

Drop methods provided in the <TableTypeIdentifier> type.

Definition

```
CREATE PROCEDURE HS_DropHistoryTableTypeMethod
    (IN TableName CHARACTER VARYING(<TableNameLength>))
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;
      DECLARE tmpTableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);
      DECLARE SpecNames CLOB ARRAY;
      DECLARE i INTEGER;

      SET tmpTableTypeIdentifier = HS_ConstructTableTypeIdentifier(TableName);

      --
      -- Drop all methods, which the routine body is defined,
      -- of <tmpTableNameIdentifier> type
      -- by using specific name of the method.
      --
      SET SpecNames = ARRAY(
            SELECT SPECIFIC_CATALOG ||
               '.' || SPECIFIC_SCHEMA ||
               '.' || SPECIFIC_NAME
            FROM INFORMATION_SCHEMA.METHOD_SPECIFICATIONS
            WHERE UDT_CATALOG = <CatalogName>
               AND UDT_SCHEMA = <SchemaName>
               AND UDT_NAME = tmpTableTypeIdentifier
```

```
                  AND ROUTINE_DEFINITION IS NOT NULL);
         SET i = 1;
         WHILE i <= CARDINALITY(SpecNames) DO
            SET stmt = 'DROP SPECIFIC METHOD ' || SpecNames[i];
            EXECUTE IMMEDIATE stmt;
            SET i = i + 1;
         END WHILE;
      END
```

Definitional Rules

1) <*CatalogName*> is <catalog name> of the schema in which <*TableNameIdentifier*> type is created.

2) <*SchemaName*> is <unqualified schema name> of the schema in which <*TableNameIdentifier*> type is created.

Description

1) The *HS_DropHistoryTableTypeMethod(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*.

5.2.4 HS_DropHistoryTrigger Procedure

Purpose

Drop history table maintenance triggers.

Definition

```
   CREATE PROCEDURE HS_DropHistoryTrigger
      (IN TableName CHARACTER VARYING(<TableNameLength>))
      LANGUAGE SQL
      DETERMINISTIC
      MODIFIES SQL DATA
      BEGIN
         DECLARE stmt CLOB;

         --
         -- Drop DELETE trigger if exist
         --
         IF EXISTS (
               SELECT *
               FROM INFORMATION_SCHEMA.TRIGGERS
               WHERE TRIGGER_CATALOG = <CatalogName>
                  AND TRIGGER_SCHEMA = <SchemaName>
                  AND TRIGGER_NAME = HS_ConstructDelTriggerIdentifier(TableName)
               ) THEN
            SET stmt =
               'DROP TRIGGER ' || HS_ConstructDelTriggerIdentifier(TableName)  ;
            EXECUTE IMMEDIATE stmt;
         END IF;

         --
         -- Drop UPDATE trigger if exist
         --
```

```
        IF EXISTS (
              SELECT *
              FROM INFORMATION_SCHEMA.TRIGGERS
              WHERE TRIGGER_CATALOG = <CatalogName>
                 AND TRIGGER_SCHEMA = <SchemaName>
                 AND TRIGGER_NAME = HS_ConstructUpdTriggerIdentifier(TableName)
              ) THEN
           SET stmt =
              'DROP TRIGGER ' || HS_ConstructUpdTriggerIdentifier(TableName)  ;
           EXECUTE IMMEDIATE stmt;
        END IF;

        --
        -- Drop INSERT trigger if exist
        --
        IF EXISTS (
              SELECT *
              FROM INFORMATION_SCHEMA.TRIGGERS
              WHERE TRIGGER_CATALOG = <CatalogName>
                 AND TRIGGER_SCHEMA = <SchemaName>
                 AND TRIGGER_NAME = HS_ConstructInsTriggerIdentifier(TableName)
              ) THEN
           SET stmt =
              'DROP TRIGGER ' || HS_ConstructInsTriggerIdentifier(TableName)   ;
           EXECUTE IMMEDIATE stmt;
        END IF;
     END
```

Definitional Rules

1) *<CatalogName>* is <catalog name> of the schema in which history table maintenance triggers are created.

2) *<SchemaName>* is <unqualified schema name> of the schema in which history table maintenance triggers are created.

Description

1) The *HS_DropHistoryTrigger(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*.

5.2.5 HS_DropHistoryTable Procedure

Purpose

Drop the history table corresponding to the specified tracked table.

Definition

```
  CREATE PROCEDURE HS_DropHistoryTable
     (IN TableName CHARACTER VARYING(<TableNameLength>))
     LANGUAGE SQL
     DETERMINISTIC
     MODIFIES SQL DATA
     BEGIN
```

```
        DECLARE stmt CLOB;

        -- Drop the history table if exists.
        IF EXISTS (
              SELECT *
              FROM INFORMATION_SCHEMA.TABLES
              WHERE TABLE_CATALOG = <CatalogName>
                 AND TABLE_SCHEMA = <SchemaName>
                 AND TABLE_NAME = HS_ConstructTableIdentifier(TableName)
              ) THEN
           SET stmt =
              'DROP TABLE ' || HS_ConstructTableIdentifier(TableName)         ;
           EXECUTE IMMEDIATE stmt;
        END IF;
     END
```

Definitional Rules

1) *<CatalogName>* is <catalog name> of the schema in which history table is created.

2) *<SchemaName>* is <unqualified schema name> of the schema in which history table is created.

Description

1) The *HS_DropHistoryTable(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*.

5.2.6 HS_DropHistoryTableType Procedure

Purpose

Drop the <TableTypeIdentifier> type.

Definition

```
  CREATE PROCEDURE HS_DropHistoryTableType
     (IN TableName CHARACTER VARYING(<TableNameLength>))
     LANGUAGE SQL
     DETERMINISTIC
     MODIFIES SQL DATA
     BEGIN
        DECLARE stmt CLOB;

        -- Drop the <TableTypeIdentifier> type if exists.
        IF EXISTS (
              SELECT *
              FROM INFORMATION_SCHEMA.USER_DEFINED_TYPES
              WHERE USER_DEFINED_TYPE_CATALOG = <CatalogName>
                 AND USER_DEFINED_TYPE_SCHEMA = <SchemaName>
                 AND USER_DEFINED_TYPE_NAME =
                    HS_ConstructTableTypeIdentifier(TableName)
              ) THEN
           SET stmt =
              'DROP TYPE ' || HS_ConstructTableTypeIdentifier(TableName)       ;
           EXECUTE IMMEDIATE stmt;
```

```
      END IF;
   END
```

Definitional Rules

1) <*CatalogName*> is <catalog name> of the schema in which <*TableTypeIdentifier*> type is created.

2) <*SchemaName*> is <unqualified schema name> of the schema in which <*TableTypeIdentifier*> type is created.

Description

1) The *HS_DropHistoryTableType(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*.

5.2.7 HS_DropHistoryTableSequenceNumberGenerator procedure

Purpose

Drop the sequence generator for generating a sequence number for a history row of a history table.

Definition

```
CREATE PROCEDURE HS_DropHistoryTableSequenceNumberGenerator
   (IN TableName CHARACTER VARYING(<TableNameLength>))
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      --
      -- Drop sequence generator if exist
      --
      IF EXISTS (
            SELECT *
            FROM INFORMATION_SCHEMA.SEQUENCES
            WHERE SEQUENCE_CATALOG = <CatalogName>
               AND SEQUENCE_SCHEMA = <SchemaName>
               AND SEQUENCE_NAME =
                     HS_ConstructSequenceGeneratorIdentifier(TableName)
            ) THEN
         EXECUTE IMMEDIATE stmt;
         SET stmt =
            'DROP SEQUENCE '                                              ||
               HS_ConstructSequenceGeneratorIdentifier(TableName)         ;
         EXECUTE IMMEDIATE stmt;
      END IF;
   END
```

Definitional Rules

1) *<CatalogName>* is <catalog name> of the schema which the sequence generator is created in.

2) *<SchemaName>* is <unqualified schema name> of the schema which the sequence generator is created in.

Description

1) The *HS_CreateHistoryTableSequenceNumberGenerator(CHARACTER VARYING(<TableNameLength>))* procedure takes the following input parameter:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*.

5.3 Utility Procedures for History

The procedures and functions in this subclause are used in common in procedures of the HS_CreateHistory procedure or the HS_DropHistory procedure.

5.3.1 Functions for extracting an identifier

Purpose

Extract an Identifier body from a character string type value that forms a schema-qualified name.

Definition

```
CREATE FUNCTION HS_ExtractCatalogIdentifier
    (TableName CHARACTER VARYING(<TableNameLength>))
    RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT
    BEGIN
        --
        -- See Description
        --
    END
CREATE FUNCTION HS_ExtractSchemaIdentifier
    (TableName CHARACTER VARYING(<TableNameLength>))
    RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT
    BEGIN
        --
        -- See Description
        --
    END
CREATE FUNCTION HS_ExtractTableIdentifier
    (TableName CHARACTER VARYING(<TableNameLength>))
    RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
```

```
      RETURNS NULL ON NULL INPUT
      BEGIN
         --
         -- See Description
         --
      END
   CREATE FUNCTION HS_ExtractColumnIdentifier
      (ColumnName CHARACTER VARYING(<ColumnNameLength>))
      RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         --
         -- See Description
         --
      END
```

Definitional Rules

1) *<IdentifierBodyLength>* is the implementation-defined maximum length of <identifier body> , <delimited identifier body>, and <Unicode delimiter body> which are defined in ISO/IEC 9075-2.

Description

1) The function *HS_ExtractCatalogIdentifier(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

   a) a CHARACTER VARYING value *TableName*.

2) For the function *HS_ExtractCatalogIdentifier(CHARACTER VARYING(<TableNameLength>))*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

   b) If the <table name> represented by the value of parameter *TableName* contains a <catalog name> *ECN*, then let *CN* be *ECN*. Otherwise, let *CN* be the <catalog name> contained in the <schema name> that is the default schema name in the SQL-session.

   Case:

   i) If *CN* is <regular identifier>, then let *RV* be the character string value that is the equivalent case-normal form of a representation of <identifier body> of *CN*.

   ii) Otherwise, let *RV* be the character string value that is a representation of <delimited identifier body>, or <Unicode delimiter body> of *CN* in the character set of SQL_IDENTIFIER.

3) The function *HS_ExtractSchemaIdentifier(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

   a) a CHARACTER VARYING value *TableName*.

4) For the function *HS_ExtractSchemaIdentifier(CHARACTER VARYING(<TableNameLength>))*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

b) If the <table name> represented by the value of parameter *TableName* contains a <local or schema qualifier> *ELS*, then let *SN* be the <unqualified schema name> contained in *ELS*. Otherwise, let *SN* be the <unqualified schema name> contained in the <schema name> that is the default schema name in the SQL-session.

Case:

i) If *SN* is <regular identifier>, then let *RV* be the character string value that is the equivalent case-normal form of a representation of <identifier body> of *SN*.

ii) Otherwise, let *RV* be the character string value that is a representation of <delimited identifier body>, or <Unicode delimiter body> of *SN* in the character set of SQL_IDENTIFIER.

5) The function *HS_ExtractTableIdentifier(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

a) a CHARACTER VARYING value *TableName*.

6) For the function *HS_ExtractTableIdentifier(CHARACTER VARYING(<TableNameLength>))*:

a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

b) Let *TN* be the <qualified identifier> contained in the <table name> represented by the value of parameter *TableName*.

Case:

i) If *TN* is <regular identifier>, then let *RV* be the character string value that is the equivalent case-normal form of a representation of <identifier body> of *TN*.

ii) Otherwise, let *RV* be the character string value that is a representation of <delimited identifier body>, or <Unicode delimiter body> of *TN* in the character set of SQL_IDENTIFIER.

7) The function *HS_ExtractColumnIdentifier(CHARACTER VARYING(<ColumnNameLength>))* takes the following input parameter:

a) a CHARACTER VARYING value *ColumnName*.

8) For the function *HS_ExtractColumnIdentifier(CHARACTER VARYING(<ColumnNameLength>))*:

a) The value of parameter *ColumnName* is a character representation of a column name which forms an instance of <column name>.

b) Let *CN* be the <column name> represented by the value of parameter *ColumnName*.

Case:

i) If *CN* is <regular identifier>, then let *RV* be the character string value that is the equivalent case-normal form of a representation of <identifier body> of *CN*.

ii) Otherwise, let *RV* be the character string value that is a representation of <delimited identifier body>, or <Unicode delimiter body> of *CN* in the character set of SQL_IDENTIFIER.

9) Return the character string whose value is the character sequence:

RV

5.3.2 HS_CreateCommaSeparatedTrackedColumnList Procedure

Purpose

For the specified tracked columns, generate comma-separated list of the texts, each of which concatenates the specified prefix and the name of the column specified.

Definition

```
CREATE PROCEDURE HS_CreateCommaSeparatedTrackedColumnList
    (IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY,
     IN ColumnNamePrefix CLOB,
     OUT ResultValue CLOB)
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE i INTEGER;

      -- Generate comma-separated list of tracked column names
      --                                              with specified prefix.

      -- Set control variable for WHILE loop to initial value 1.
      SET i = 1;
      -- Initially set output parameter to empty string.
      SET ResultValue = '';
      -- Repeat while i is less than or equal to the number of tracked
      columns.
      WHILE i <= CARDINALITY(TrackedColumns) DO
         -- In case i > 1, that is each time except first time of loop,
         IF i > 1 THEN
            -- Concatenate comma character into output parameter.
            SET ResultValue = ResultValue || ', '                          ;
         END IF;
         -- In case the specified prefix is applicable,
         IF ColumnNamePrefix IS NOT NULL THEN
            -- Concatenate the specified prefix into output parameter.
            SET ResultValue = ResultValue || ColumnNamePrefix              ;
         END IF;
         -- Concatenate the i-th tracked column name into output parameter.
         SET ResultValue = ResultValue ||
            HS_ConstructColumnIdentifier(TrackedColumns[i])                ;
         -- Increment control variable for WHILE loop.
         SET i = i + 1;
      END WHILE;
   END
```

Description

1) The *HS_CreateCommaSeparatedTrackedColumnList(CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*,

   b) a CLOB value *ColumnNamePrefix*.

2) The *HS_CreateCommaSeparatedTrackedColumnList(CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

3) For the procedure *HS_CreateCommaSeparatedTrackedColumnList(CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB)*:

   a) The value of the element of the array of parameter *TrackedColumns* is a character representation of a column name which forms an instance of <column name>.

   b) Let *CNn* be the <column name> represented by the n-th element of array of parameter *TrackedColumns*.

   c) Case:

      i) If the value of parameter *ColumnNamePrefix* is not null value, then let *PRX* be the value of parameter *ColumnNamePrefix*.

      ii) Otherwise, let *PRX* be the character string of length 0(zero).

   d) Let *CIDn* be the value of the result of *HS_ConstructColumnIdentifier(CNn)*. Let *PIDn* be the character string value of a concatenation:

      <quote>*PRX*<quote> || *CIDn*

   e) Let *RV* be the character string value of a concatenation:

      PID1 || <quote><comma><quote> || *PID2* || <quote><comma><quote> || ... || *PIDn*

   f) Set the value of output parameter *ResultValue* to *RV*.

5.3.3 HS_CreateCommaSeparatedTrackedColumnAndTypeList Procedure

Purpose

For all tracked columns of the specified table, generate comma-separated list of the texts, each of which concatenates the specified prefix, the name of the tracked column, white space and the name of a data type or the name of a domain.

Definition

```
CREATE PROCEDURE HS_CreateCommaSeparatedTrackedColumnAndTypeList
    (IN TableName CHARACTER VARYING(<TableNameLength>),
     IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY,
     OUT ResultValue CLOB)
    LANGUAGE SQL
    DETERMINISTIC
    MODIFIES SQL DATA
    BEGIN
        DECLARE TrackedColumnsAndTypes CLOB ARRAY;

        DECLARE i INTEGER;

        -- Generate comma-separated list of pairs of
        -- tracked column name with specified prefix and data type or domain
        name.

        --
        -- !! See Description
        --        about "TrackedColumnsAndTypes"
        --
```

```
      -- Set control variable for WHILE loop to initial value 1.
      SET i = 1;
      -- Initially set output parameter to empty string.
      SET ResultValue = '';
      -- Repeat while i is less than or equal to the number of tracked
      columns.
      WHILE i <= CARDINALITY(TrackedColumnsAndTypes) DO
         -- In case i > 1, that is each time except first time of loop,
         IF i > 1 THEN
            -- Concatenate comma character into output parameter.
            SET ResultValue = ResultValue || ', '                     ;
         END IF;
         -- Concatenate the i-th pair of tracked column name with prefix
         --                  and data type or domain name into output
         parameter.
         SET ResultValue = ResultValue || TrackedColumnsAndTypes[i]        ;
         -- Increment control variable for WHILE loop.
         SET i = i + 1;
      END WHILE;
   END
```

Description

1) The *HS_CreateCommaSeparatedTrackedColumnAndTypeList(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

2) The *HS_CreateCommaSeparatedTrackedColumnAndTypeList(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

3) For the procedure *HS_CreateCommaSeparatedTrackedColumnAndTypeList(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB)*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

   b) The value of the element of the array of parameter *TrackedColumns* is a character representation of a column name which forms an instance of <column name>.

   c) Let *TN* be the <qualified identifier> contained in the <table name> represented by the value of parameter *TableName*.

   d) Let *CNn* be the <column name> represented by the n-th element of the array of parameter *TrackedColumns*.

   c) Let *CIDn* be the value of the result of *HS_ConstructColumnIdentifier(CNn)*.

   d) Let *DTn* be the <data type or domain name> which corresponds to the column *CNn*.

   e) Let *CDNn* be the character string value of a concatenation:

      CIDn || <quote><space><quote> || <quote>*DTn*<quote>

f) Set the value of n-th element of array *TrackedColumnsAndTypes* to *CDNn*.

g) Let *RV* be the character string value of a concatenation:

CDN1 || <quote><comma><quote> || *CDN2* || <quote><comma><quote> || ... || *CDNn*

h) Set the value of output parameter *ResultValue* to *RV*.

5.3.4 Functions for constructing an identifier and <IdentifierLength>

Purpose

Provide functions for constructing an Identifier from a character string type value that forms a table name or a column name.

Definition

```
CREATE FUNCTION HS_ConstructIdentifier
    (Prefix CHARACTER VARYING(<PrefixOrPostfixLength>),
     SourceName CHARACTER VARYING(<TableOrColumnNameLength>),
     Postfix CHARACTER VARYING(<PrefixOrPostfixLength>))
   RETURNS CHARACTER VARYING(<IdentifierLength>)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      --
      -- See Description
      --
   END
CREATE FUNCTION HS_ConstructTableIdentifier
    (TableName CHARACTER VARYING(<TableNameLength>))
   RETURNS CHARACTER VARYING(<IdentifierLength>)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      -- Return the identifier extracted from specified table name,
      --                                  and concatenated with prefix HS_TBL_.
      RETURN HS_ConstructIdentifier('HS_TBL_', TableName, '');
   END
CREATE FUNCTION HS_ConstructTableTypeIdentifier
    (TableName CHARACTER VARYING(<TableNameLength>))
   RETURNS CHARACTER VARYING(<IdentifierLength>)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   BEGIN
      -- Return the identifier extracted from specified table name,
      --                                  and concatenated with prefix HS_TYPE_.
      RETURN HS_ConstructIdentifier('HS_TYPE_', TableName, '');
   END
CREATE FUNCTION HS_ConstructColumnIdentifier
    (ColumnName CHARACTER VARYING(<ColumnNameLength>))
   RETURNS CHARACTER VARYING(<IdentifierLength>)
```

```
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         -- Return the identifier extracted from specified column name.
         RETURN HS_ConstructIdentifier('', ColumnName, '');
      END
   CREATE FUNCTION HS_ConstructInsTriggerIdentifier
      (TableName CHARACTER VARYING(<TableNameLength>))
      RETURNS CHARACTER VARYING(<IdentifierLength>)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         -- Return the identifier extracted from specified table name,
         --             and concatenated with prefix HS_TR_ and postfix _INS.
         RETURN HS_ConstructIdentifier('HS_TR_', TableName, '_INS');
      END
   CREATE FUNCTION HS_ConstructUpdTriggerIdentifier
      (TableName CHARACTER VARYING(<TableNameLength>))
      RETURNS CHARACTER VARYING(<IdentifierLength>)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         -- Return the identifier extracted from specified table name,
         --             and concatenated with prefix HS_TR_ and postfix _UPD.
         RETURN HS_ConstructIdentifier('HS_TR_', TableName, '_UPD');
      END
   CREATE FUNCTION HS_ConstructDelTriggerIdentifier
      (TableName CHARACTER VARYING(<TableNameLength>))
      RETURNS CHARACTER VARYING(<IdentifierLength>)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         -- Return the identifier extracted from specified table name,
         --             and concatenated with prefix HS_TR_ and postfix _DEL.
         RETURN HS_ConstructIdentifier('HS_TR_', TableName, '_DEL');
      END
   CREATE FUNCTION HS_ConstructSequenceGeneratorIdentifier
      (TableName CHARACTER VARYING(<TableNameLength>))
      RETURNS CHARACTER VARYING(<IdentifierLength>)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         -- Return the identifier extracted from specified table name,
         --             and concatenated with prefix HS_SEQ_.
         RETURN HS_ConstructIdentifier('HS_SEQ_', TableName, '');
      END
```

Definitional Rules

1) *<PrefixOrPostfixLength>* is the length 10.

2) *<TableOrColumnNameLength>* is the maximum length of *<TableNameLength>* and *<ColumnNameLength>*.

3) *<IdentifierLength>* is the implementation-defined maximum length of <regular identifier>, <delimited identifier>, and <Unicode delimited identifier> which are defined in ISO/IEC 9075-2.

Description

1) For the function *HS_ConstructIdentifier(Prefix CHARACTER VARYING(<PrefixOrPostfixLength>), SourceName CHARACTER VARYING(<TableOrColumnNameLength>), Postfix CHARACTER VARYING(<PrefixOrPostfixLength>))*:

   a) Let *PRX* be the value of parameter *Prefix*. Let *SNM* be the value of parameter *SourceName*.

      Case:

      i) If *PRX* is a character string of length 0(zero), then let *IDB* be the value of the result of *HS_ExtractColumnIdentifier(SNM)*.

      ii) Otherwise, let *IDB* be the value of the result of *HS_ExtractTableIdentifier(SNM)*.

   b) Let *PTX* be the value of parameter *Postfix*. Let *PID* be the character string value of a concatenation:

      <quote>*PRX*<quote> || *IDB* || <quote>*PTX*<quote>

   c) Case:

      i) If *SNM* constitutes <Unicode delimited identifier>, then let *ESC* be the specified or implied <Unicode escape specifier>. Return the character string whose value is the character sequence of <Unicode delimited identifier>:

         U&<double quote>*PID*<double quote> *ESC*

      ii) If *SNM* constitutes <delimited identifier>, then return the character string whose value is the character sequence of <delimited identifier>:

         <double quote>*PID*<double quote>.

      iii) Otherwise, return the character string whose value is the character sequence of <regular identifier>:

         PID.

2) For the function *HS_ConstructTableIdentifier(CHARACTER VARYING(<TableNameLength>))*, the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

3) For the function *HS_ConstructTableTypeIdentifier(CHARACTER VARYING(<TableNameLength>))*, the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

4) For the function *HS_ConstructColumnIdentifier(CHARACTER VARYING(<ColumnNameLength>))*, the value of parameter *ColumnName* is a character representation of a column name which forms an instance of <column name>.

5) For the function *HS_ConstructInsTriggerIdentifier(CHARACTER VARYING(<TableNameLength>))*, the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

6) For the function *HS_ConstructUpdTriggerIdentifier(CHARACTER VARYING(<TableNameLength>))*, the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

7) For the function *HS_ConstructDelTriggerIdentifier(CHARACTER VARYING(<TableNameLength>))*, the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

8) For the function *HS_ConstructSequenceGeneratorIdentifier(CHARACTER VARYING(<TableNameLength>))*, the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

5.3.5 HS_GetPrimaryKeys function

Purpose

Obtain the primary key columns of the specified table.

Definition

```
CREATE FUNCTION HS_GetPrimaryKeys
   (TableName CHARACTER VARYING(<TableNameLength>))
   RETURNS TABLE(PrimaryKeyColumn CHARACTER VARYING(<ColumnNameLength>))
   LANGUAGE SQL
   DETERMINISTIC
   READS SQL DATA
   BEGIN
      --
      -- This function requires only columns whose constraint type
      -- is 'PRIMARY KEY', but the type of constraint is not contained in
      -- KEY_COLUMN_USAGE view.  This is contained in TABLE_CONSTRAINTS view.
      -- Therefore join these views and obtain only the primary keys.
      --
      RETURN TABLE(
         SELECT KCU.COLUMN_NAME
            FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KCU
               INNER JOIN
               INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
            ON KCU.CONSTRAINT_CATALOG = TC.CONSTRAINT_CATALOG
               AND KCU.CONSTRAINT_SCHEMA = TC.CONSTRAINT_SCHEMA
               AND KCU.CONSTRAINT_NAME = TC.CONSTRAINT_NAME
               AND KCU.TABLE_CATALOG = TC.TABLE_CATALOG
               AND KCU.TABLE_SCHEMA = TC.TABLE_SCHEMA
               AND KCU.TABLE_NAME = TC.TABLE_NAME
            WHERE KCU.TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
               AND KCU.TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
               AND KCU.TABLE_NAME = HS_ExtractTableIdentifier(TableName)
               AND TC.CONSTRAINT_TYPE = 'PRIMARY KEY'
            ORDER BY ORDINAL_POSITION);
   END
```

Description

1) The function *HS_GetPrimaryKeys(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

   a) a CHARACTER VARYING value *TableName*.

2) For the function *HS_GetPrimaryKeys(CHARACTER VARYING(<TableNameLength>))*:

a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

b) Return a table that contains a single column *PrimaryKeyColumn* of the column names of the primary key of <table name>.

5.3.6 HS_GetTransactionTimestamp function

Purpose

Obtain a timestamp value for an SQL-transaction.

Definition

```
CREATE FUNCTION HS_GetTransactionTimestamp()
   RETURNS TIMESTAMP(<TimestampPrecision>)
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      --
      -- !! See Description
      --
   END
```

Description

1) The *HS_GetTransactionTimestamp()* function has no input parameters.

2) For the *HS_GetTransactionTimestamp()* function:

   a) Returns the value of transaction timestamp for an SQL-transaction.

5.3.7 HS_GetHistoryRowSetIdentifierColumns procedure

Purpose

For the specified tracked table, obtain the identifier columns of history row set, which are the unique constraint columns with NOT NULL of the tracked table.

Definition

```
CREATE PROCEDURE HS_GetHistoryRowSetIdentifierColumns
   (IN TableName CHARACTER VARYING(<TableNameLength>),
    IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY,
    OUT IdentifierColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
   LANGUAGE SQL
   DETERMINISTIC
   READS SQL DATA
   BEGIN
      DECLARE AllPKeyColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY;

      DECLARE ConstNames CHARACTER VARYING(<ConstraintNameLength>) ARRAY;
      DECLARE ColumnNames CHARACTER VARYING(<ColumnNameLength>) ARRAY;

      DECLARE i INTEGER;
DECLARE j INTEGER;
DECLARE found INTEGER;

-- Get primary key columns of the specified table.
SET AllPKeyColumns = ARRAY(TABLE (HS_GetPrimaryKeys(TableName)));

--
-- If specified tracked table has primary key
--  and all columns of primary key are specified
--  in the input parameter TrackedColumns,
--  then return the primary key columns through
--  the output parameter IdentifierColumns.
-- Otherwise, try to find an unique constraint,
--  all columns of which have NOT NULL constraints
--  and are specified in the input parameter TrackedColumns.
--   If any available unique constraint is found,
--    then return the unique key columns through
--    the output parameter IdentifierColumns.
--   Otherwise, the output parameter IdentifierColumns is null.
--
IF EXISTS(SELECT * FROM UNNEST(AllPKeyColumns))
      AND NOT EXISTS(SELECT *
         FROM UNNEST(AllPKeyColumns) AS UC(COL)
         WHERE COL NOT IN (
            SELECT COL FROM UNNEST(TrackedColumns) AS TC(COL)) THEN
   SET IdentifierColumns = AllPKeyColumns;
ELSE
   --
   -- Obtain the constraint names of the unique constraints
   -- of the specified tracked table.
   --
   SET ConstNames = ARRAY(
         SELECT CONSTRAINT_CATALOG ||
            '.' || CONSTRAINT_SCHEMA ||
            '.' || CONSTRAINT_NAME
         FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
         WHERE
            TABLE_CATALOG = HS_ExtractCatalogIdentifier(TableName)
            AND TABLE_SCHEMA = HS_ExtractSchemaIdentifier(TableName)
            AND TABLE_NAME = HS_ExtractTableIdentifier(TableName)
            AND CONSTRAINT_TYPE = 'UNIQUE');
   --
   -- For each unique constraint of the specified tracked table,
   --  check the followings:
   --   1. All columns of the part of unique constraint are not
nullable.
   --   2. All columns of the part of unique constraint are specified
   --      in the parameter TrackedColumns.
   -- If above mentioned unique constraint is found,
   --  return the column names of the unique constraint
   --  through the output parameter IdentifierColumns.
```

```
          --
          -- Set control variable for WHILE loop to initial value 1.
          SET i = 1;
          -- Initially set ”found” flag to 0(not found).
          SET found = 0;
          -- Repeat while available unique constraint is not found and
          --       i is less than or equal to the number of unique constraints.
          WHILE found = 0 AND i <= CARDINALITY(ConstNames) DO
             -- In case i-th unique constraint has no nullable columns,
             IF NOT EXISTS(SELECT *
                      FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KCU
                         INNER JOIN INFORMATION_SCHEMA.COLUMNS AS C
                         ON KCU.TABLE_CATALOG = C.TABLE_CATALOG
                            AND KCU.TABLE_SCHEMA = C.TABLE_SCHEMA
                            AND KCU.TABLE_NAME = C.TABLE_NAME
                            AND KCU.COLUMN_NAME = C.COLUMN_NAME
                      WHERE IS_NULLABLE = 'YES'
                         AND CONSTRAINT_CATALOG ||
                         '.' || CONSTRAINT_SCHEMA ||
                         '.' || CONSTRAINT_NAME = ConstNames[i]) THEN
                -- Get column names of i-th unique constraint.
                SET ColumnNames = ARRAY(
                      SELECT COLUMN_NAME
                      FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
                      WHERE CONSTRAINT_CATALOG ||
                         '.' || CONSTRAINT_SCHEMA ||
                         '.' || CONSTRAINT_NAME = ConstNames[i]);
                -- In case every column of i-th unique constraint is
                --                                   any of tracked columns.
                IF NOT EXISTS(SELECT *
                      FROM UNNEST(ColumnNames) AS UC(COL)
                      WHERE COL NOT IN (
                         SELECT  COL  FROM  UNNEST(TrackedColumns)  AS  TC(COL)))
                         THEN
                   -- Set output parameter to column names of
                   --                                  i-th unique constraint;
                   SET IdentifierColumns = ColumnNames;
                   SET found = 1;
                END IF;
             END IF;
             -- Increment control variable for WHILE loop.
             SET i = i + 1;
          END WHILE;
       END IF;
    END
```

Definitional Rules

1) <ConstraintNameLength> is the implementation-defined maximum length of <constraint name> which are defined in ISO/IEC 9075-2.

Description

1) The *HS_GetHistoryRowSetIdentifierColumns(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

b) a CHARACTER VARYING(<*ColumnNameLength*>) value *TrackedColumns*.

2) The *HS_GetHistoryRowSetIdentifierColumns(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CHARACTER VARYING(<ColumnNameLength>) ARRAY)* procedure takes the following output parameter:

a) a CHARACTER VARYING(<ColumnNameLength>) value IdentifierColumns.

3) For the procedure *HS_GetHistoryRowSetIdentifierColumns(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CHARACTER VARYING(<ColumnNameLength>) ARRAY)*:

a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

b) The value of each element of the array of parameter *TrackedColumns* is a character representation of a column name which forms an instance of <column name>.

5.3.8 HS_CreateCommaSeparatedIdentifierColumnList procedure

Purpose

For the identifier columns of history row set of the specified table, which are the unique constraint columns with NOT NULL of the tracked table, generate comma-separated list of the texts, each of which concatenates the name of the identifier column and the specified postfix.

Definition

```
CREATE PROCEDURE HS_CreateCommaSeparatedIdentifierColumnList
   (IN TableName CHARACTER VARYING(<TableNameLength>),
    IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY,
    IN ColumnNamePostfix CLOB,
    OUT ResultValue CLOB)
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE IdentifierColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY;

      DECLARE i INTEGER;

      -- Get identifier columns of history row set of the specified table.
      CALL HS_GetHistoryRowSetIdentifierColumns(
         TableName, TrackedColumns, IdentifierColumns);

      -- Generate comma-separated list of
      --                          identifier column names with specified
      postfix.

      -- Set control variable for WHILE loop to initial value 1.
      SET i = 1;
      -- Initially set output parameter to empty string.
      SET ResultValue = '';
      -- Repeat while i is less than or equal to the number of identifier
      columns.
      WHILE i <= CARDINALITY(IdentifierColumns) DO
         -- In case i > 1, that is each time except first time of loop,
         IF i > 1 THEN
```

```
            -- Concatenate comma character into output parameter.
            SET ResultValue = ResultValue || ', ' ;
         END IF;
         -- Concatenate the i-th identifier column name into output parameter.
         SET ResultValue = ResultValue ||
            HS_ConstructColumnIdentifier(IdentifierColumns[i]) ;
         -- In case the specified postfix is applicable,
         IF ColumnNamePostfix IS NOT NULL THEN
            -- Concatenate the specified postfix into output parameter.
            SET ResultValue = ResultValue || ColumnNamePostfix ;
         END IF;
         -- Increment control variable for WHILE loop.
         SET i = i + 1;
      END WHILE;
   END
```

Description

1) The *HS_CreateCommaSeparatedIdentifierColumnList(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*,

   c) a CLOB value *ColumnNamePostfix*.

2) The *HS_CreateCommaSeparatedIdentifierColumnList(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

3) For the procedure *HS_CreateCommaSeparatedIdentifierColumnList(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB)*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

   b) The value of the element of the array of parameter *TrackedColumns* is a character representation of a column name which forms an instance of <column name>.

   c) Case:

      i) If the value of parameter *ColumnNamePostfix* is not null value, then let *PTX* be the value of parameter *ColumnNamePostfix*.

      ii) Otherwise, let *PTX* be the character string of length 0(zero).

   d) Let *CNn* be the <column name> represented by the n-th element of array of the output parameter *IdentifierColumns* of the invocation of procedure:

      HS_GetHistoryRowSetIdentifierColumns(TableName, TrackedColumns, IdentifierColumns).

   e) Let *CIDn* be the value of the result of *HS_ConstructColumnIdentifier(CNn)*. Let *PIDn* be the character string value:

CIDn || <quote>*PTX*<quote>

f) Let *RV* be the character string value of a concatenation:

PID1 || <quote><comma><quote> || *PID2* || <quote><comma><quote> || ... || *PIDn*

g) Set the value of output parameter *ResultValue* to *RV*.

5.3.9 HS_CreateIdentifierColumnSelfJoinCondition procedure

Purpose

For the identifier columns of history row set of the specified table, which are the unique constraint columns with NOT NULL of the specified table, generate the text of self-join condition for the specified table. The text is a list of equals comparison predicates separated by ' AND '. Each equal predicate is a text which concatenates the specified prefix text, the name of column, an equals operator, another specified prefix text and the name of column.

Definition

```
CREATE PROCEDURE HS_CreateIdentifierColumnSelfJoinCondition
    (IN TableName CHARACTER VARYING(<TableNameLength>),
     IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>),
     IN ColumnNamePrefix1 CLOB,
     IN ColumnNamePrefix2 CLOB,
     OUT ResultValue CLOB)
    LANGUAGE SQL
    DETERMINISTIC
    MODIFIES SQL DATA
    BEGIN
       DECLARE IdentifierColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY;

       DECLARE i INTEGER;

       -- Get the identifier columns of history row set of the specified table.
       CALL HS_GetHistoryRowSetIdentifierColumns(
          TableName, TrackedColumns, IdentifierColumns);

       -- Set control variable for WHILE loop to initial value 1.
       SET i = 1;
       -- Initially set output parameter to empty string.
       SET ResultValue = '';
       -- Repeat while i is less than or equal to the number of identifier
       columns.
       WHILE i <= CARDINALITY(IdentifierColumns) DO
          -- In case i > 1, that is each time except first time of loop,
          IF i > 1 THEN
             -- Concatenate AND boolean operator into output parameter.
             SET ResultValue = ResultValue || ' AND ' ;
          END IF;
          -- In case the prefix specified by ColumnNamePrefix1 is applicable,
          IF ColumnNamePrefix1 IS NOT NULL THEN
             -- Concatenate the prefix in ColumnNamePrefix1 into output
             parameter.
             SET ResultValue = ResultValue || ColumnNamePrefix1 ;
          END IF;
          -- Concatenate the i-th identifier column name into output parameter.
          SET ResultValue = ResultValue ||
               HS_ConstructColumnIdentifier(IdentifierColumns[i]) ;

            -- Concatenate equals operator into output parameter.
            SET ResultValue = ResultValue || ' = ' ;

            -- In case the prefix specified by ColumnNamePrefix2 is applicable,
            IF ColumnNamePrefix2 IS NOT NULL THEN
               --   Concatenate   the   prefix   in   ColumnNamePrefix2   into   output
               parameter.
               SET ResultValue = ResultValue || ColumnNamePrefix2 ;
            END IF;
            -- Concatenate the i-th identifier column name into output parameter.
            SET ResultValue = ResultValue ||
               HS_ConstructColumnIdentifier(IdentifierColumns[i]) ;

            -- Increment control variable for WHILE loop.
            SET i = i + 1;
         END WHILE;
      END
```

Description

1) The *HS_CreateIdentifierColumnSelfJoinCondition(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB, CLOB)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*TableNameLength*>) value *TableName*,

   b) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*,

   c) a CLOB value *ColumnNamePrefix1*,

   d) a CLOB value *ColumnNamePrefix2*.

2) The *HS_CreateIdentifierColumnSelfJoinCondition(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

3) For the procedure *HS_CreateIdentifierColumnSelfJoinCondition(CHARACTER VARYING(<TableNameLength>), CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB, CLOB, CLOB)*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

   b) The value of each element of the array of parameter *TrackedColumns* is a character representation of a column name which forms an instance of <column name>.

   c) Case:

      i) If the value of parameter *ColumnNamePrefix1* is not null value, then let *PRX1* be the value of parameter *ColumnNamePrefix1*.

      ii) Otherwise, let *PRX1* be the character string of length 0(zero).

   d) Case:

i) If the value of parameter *ColumnNamePrefix2* is not null value, then let *PRX2* be the value of parameter *ColumnNamePrefix2*.

ii) Otherwise, let *PRX2* be the character string of length 0(zero).

e) Let *CNn* be the <column name> represented by the n-th element of array of the output parameter *IdentifierColumns* of the invocation of procedure:

HS_GetHistoryRowSetIdentifierColumns(TableName, TrackedColumns, IdentifierColumns).

f) Let *CIDn* be the value of the result of *HS_ConstructColumnIdentifier(CNn)*. Let CNDn be the character string value of a concatenation:

<quote>*PRX1*<quote> || *CIDn* || <quote><equals operator><quote> || <quote>*PRX2*<quote> || *CIDn*

g) Let *RV* be the character string value of a concatenation:

CND1 || <quote>AND<quote> || *CND2* || <quote>AND<quote> || ... || *CNDn*

h) Set the value of output parameter *ResultValue* to *RV*.

5.3.10 Functions for constructing an identifier literal

Purpose

Provide functions for constructing an Identifier listeral from a character string type value that forms a schema-qualified name.

Definition

```
CREATE FUNCTION HS_ConstructCatalogIdentifierLiteral
    (TableName CHARACTER VARYING(<TableNameLength>))
    RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT
    BEGIN
        --
        -- See Description
        --
    END
CREATE FUNCTION HS_ConstructSchemaIdentifierLiteral
    (TableName CHARACTER VARYING(<TableNameLength>))
    RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT
    BEGIN
        --
        -- See Description
        --
    END
CREATE FUNCTION HS_ConstructTableIdentifierLiteral
    (TableName CHARACTER VARYING(<TableNameLength>))
    RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
    LANGUAGE SQL
```

```
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         --
         -- See Description
         --
      END
   CREATE FUNCTION HS_ConstructColumnIdentifierLiteral
      (ColumnName CHARACTER VARYING(<ColumnNameLength>))
      RETURNS CHARACTER VARYING(<IdentifierBodyLength>)
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT
      BEGIN
         --
         -- See Description
         --
      END
```

Definitional Rules

1) *<IdentifierBodyLength>* is the implementation-defined maximum length of <identifier body> , <delimited identifier body>, and <Unicode delimiter body> which are defined in ISO/IEC 9075-2.

Description

1) The function *HS_ConstructCatalogIdentifierLiteral(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

   a) a CHARACTER VARYING value *TableName*.

2) For the function *HS_ConstructCatalogIdentifierLiteral(CHARACTER VARYING(<TableNameLength>))*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

   b) Let *TN* be the <table name> represented by the value of parameter *TableName*.

   c) Let *RV* be the value of the result of *HS_ExtractCatalogIdentifier(TN)*.

3) The function *HS_ConstructSchemaIdentifierLiteral(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

   a) a CHARACTER VARYING value *TableName*.

4) For the function *HS_ConstructSchemaIdentifierLiteral(CHARACTER VARYING(<TableNameLength>))*:

   a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

   b) Let *TN* be the <table name> represented by the value of parameter *TableName*.

   c) Let *RV* be the value of the result of *HS_ExtractSchemaIdentifier(TN)*.

5) The function *HS_ConstructTableIdentifierLiteral(CHARACTER VARYING(<TableNameLength>))* takes the following input parameter:

a) a CHARACTER VARYING value *TableName*.

6) For the function *HS_ConstructTableIdentifierLiteral(CHARACTER VARYING(<TableNameLength>))*:

a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

b) Let *TN* be the <table name> represented by the value of parameter *TableName*.

c) Let *RV* be the value of the result of *HS_ExtractTableIdentifier(TN)*.

7) The function *HS_ConstructColumnIdentifierLiteral(CHARACTER VARYING(<ColumnNameLength>))* takes the following input parameter:

a) a CHARACTER VARYING value *ColumnName*.

8) For the function *HS_ConstructColumnIdentifierLiteral(CHARACTER VARYING(<ColumnNameLength>))*:

a) The value of parameter *ColumnName* is a character representation of a column name which forms an instance of <column name>.

b) Let *CN* be the <column name> represented by the value of parameter *ColumnName*.

c) Let *RV* be the value of the result of *HS_ExtractColumnIdentifier(CN)*.

9) Return the character string whose value is the character sequence:

<quote>*RV*<quote>

5.3.11 HS_CreateCommaSeparatedTrackedColumnLiteralList procedure

Purpose

For the specified tracked columns, generate comma-separated list of the character string literal which represents each column name.

Definition

```
CREATE PROCEDURE HS_CreateCommaSeparatedTrackedColumnLiteralList
    (IN TrackedColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY,
     OUT ResultValue CLOB)
    LANGUAGE SQL
    DETERMINISTIC
    MODIFIES SQL DATA
    BEGIN
       DECLARE i INTEGER;

       -- Set control variable for WHILE loop to initial value 1.
       SET i = 1;
       -- Initially set output parameter to empty string.
       SET ResultValue = '';
       -- Repeat while i is less than or equal to the number of tracked
       columns.
       WHILE i <= CARDINALITY(TrackedColumns) DO
          -- In case i > 1, that is each time except first time of loop,
          IF i > 1 THEN
             -- Concatenate comma character into output parameter.
             SET ResultValue = ResultValue || ',';
```

```
            END IF;
            -- Concatenate the quoted string of the i-th identifier column name
            --                                                   into output parameter.
            SET ResultValue = ResultValue ||
            HS_ConstructColumnIdentifierLiteral(TrackedColumns[i]);
            -- Increment control variable for WHILE loop.
            SET i = i + 1;
        END WHILE;
    END
```

Description

1) The *HS_CreateCommaSeparatedTrackedColumnLiteralList(CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB)* procedure takes the following input parameters:

   a) a CHARACTER VARYING(<*ColumnNameLength*>) ARRAY value *TrackedColumns*.

2) The *HS_CreateCommaSeparatedTrackedColumnLiteralList(CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

3) For the procedure *HS_CreateCommaSeparatedTrackedColumnLiteralList(CHARACTER VARYING(<ColumnNameLength>) ARRAY, CLOB)*:

   a) The value of the element of the array of parameter *TrackedColumns* is a character representation of a column name which forms an instance of <column name>.

5.4 <TableNameLength> and <ColumnNameLength>

Purpose

Provide the definition of <TableNameLength> and <ColumnNameLength>.

Description

1) The maximum length *L* of variable length character string type specified in the declaration of a parameter whose value is supported to be a character representation of a table name which conforms to the Format and Syntax Rules of <table name> defined in ISO/IEC 9075-2 is implementation-defined. <*TableNameLength*> is the character representation of *L*.

2) The maximum length *L* of variable length character string type specified in the declaration of a parameter whose value is supported to be a character representation of a column name which conforms to the Format and Syntax Rules of <column name> defined in ISO/IEC 9075-2 is implementation-defined. <*ColumnNameLength*> is the character representation of *L*.

5.5 Schema for <TableTypeIdentifier> Type

Purpose

Specify the schema which <*TableTypeIdentifier*> type is created in.

Definitional Rules

1) For each schema that includes a tracked table, a schema *S* is provided in order to create the <*TableTypeIdentifier*> type. The schema *S* is created before an invocation of *HS_CreateHistory* procedure or is effectively created on an execution of *HS_CreateHistory*.

NOTE 1 Objects other than the <*TableTypeIdentifier*> type and its methods, such as a history table and triggers, may also be created in schema *S*. However, this Technical Specification requires only <*TableTypeIdentifier*> type and its methods for the conformance.

NOTE 2 Schema *S* may or may not be identical to the schema that includes the tracked table whose history is being defined.

2) <*TableName*> is <table name> that represents a name of the tracked table to which <*TableTypeIdentifier*> type corresponds.

3) <*CatalogName*> is equivalent to <catalog name> contained in <*TableName*>.

4) <*SchemaName*> is <unqualified schema name> which is distinct from <unqualified schema name>s of any other schemata included in the catalog whose name is <*CatalogName*>.

5) <catalog name> and <unqualified schema name> of the schema provided for <*TableTypeIdentifier*> type are <*CatalogName*> and <*SchemaName*>, respectively.

Description

1) The <schema name> of <*CatalogName*> . <*SchemaName*>, together with the schema which includes routines and types predefined by ISO/IEC 13249, is contained in the implementation-defined <schema name list> of the SQL-path of an SQL-client module, an SQL-server module, <schema definition>, or an SQL-session.

5.6 <TimestampPrecision>

Purpose

Provide the definition of <TimestampPrecision>.

Description

1) The timestamp precision *P* of timestamp type specified in the declaration of an attribute or a parameter whose value is supported to be a character representation of a timestamp precision which conforms to the Format and Syntax Rules of <timestamp precision> defined in ISO/IEC 9075-2 is implementation-defined. <*TimestampPrecision*> is the character representation of *P*.

6 History Types

6.1 HS_History Type and Routines

6.1.1 HS_History Type

Purpose

Provide the definition of a structured type for a period of a history row.

Definition

```
CREATE TYPE HS_History
   AS (
      HS_BeginTime TIMESTAMP(<TimestampPrecision>),
      HS_EndTime TIMESTAMP(<TimestampPrecision>)
   )

   CONSTRUCTOR METHOD HS_History
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS HS_History
      SELF AS RESULT,

   METHOD HS_Overlaps
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS INTEGER,

   METHOD HS_Overlaps
      (hs_hist HS_History)
      RETURNS INTEGER,

   METHOD HS_Meets
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS INTEGER,

   METHOD HS_Meets
      (hs_hist HS_History)
      RETURNS INTEGER,

   METHOD HS_Precedes
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS INTEGER,

   METHOD HS_Precedes
      (hs_hist HS_History)
      RETURNS INTEGER,

   METHOD HS_PrecedesOrMeets
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS INTEGER,
METHOD HS_PrecedesOrMeets
   (hs_hist HS_History)
   RETURNS INTEGER,

METHOD HS_Succeeds
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER,

METHOD HS_Succeeds
   (hs_hist HS_History)
   RETURNS INTEGER,

METHOD HS_SucceedsOrMeets
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER,

METHOD HS_SucceedsOrMeets
   (hs_hist HS_History)
   RETURNS INTEGER,

METHOD HS_Contains
   (timePoint TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER,

METHOD HS_Contains
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER,

METHOD HS_Contains
   (hs_hist HS_History)
   RETURNS INTEGER,

METHOD HS_Equals
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER,

METHOD HS_Equals
   (hs_hist HS_History)
   RETURNS INTEGER,

METHOD HS_MonthInterval()
   RETURNS INTERVAL YEAR TO MONTH,

METHOD HS_DayInterval()
   RETURNS INTERVAL DAY TO SECOND,

METHOD HS_Intersect
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS HS_History,

METHOD HS_Intersect
   (hs_hist HS_History)
   RETURNS HS_History,
METHOD HS_Union
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS HS_History,

METHOD HS_Union
   (hs_hist HS_History)
   RETURNS HS_History,

METHOD HS_Except
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS HS_History,

METHOD HS_Except
   (hs_hist HS_History)
   RETURNS HS_History
```

Description

1) The *HS_History* type provides the following methods for public use:

   a) a method HS_Overlaps(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   b) a method HS_Overlaps(HS_History),

   c) a method HS_Meets(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   d) a method HS_Meets(HS_History),

   e) a method HS_Precedes(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   f) a method HS_Precedes(HS_History),

   g) a method HS_PrecedesOrMeets(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   h) a method HS_PrecedesOrMeets(HS_History),

   i) a method HS_Succeeds(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   j) a method HS_Succeeds(HS_History),

   k) a method HS_SucceedsOrMeets(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   l) a method HS_SucceedsOrMeets(HS_History),

   m) a method HS_Contains(TIMESTAMP(<TimestampPrecision>)),

   n) a method HS_Contains(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

   o) a method HS_Contains(HS_History),

   p) a method HS_Equals(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

q) a method HS_Equals(HS_History),

r) a method HS_MonthInterval(),

s) a method HS_DayInterval(),

t) a method HS _Intersect(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

u) a method HS _Intersect(HS_History),

v) a method HS_Union(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

w) a method HS_Union(HS_History),

x) a method HS_Except(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>)),

y) a method HS_Except(HS_History).

2) The *HS_History* type has the following attributes:

a) a TIMESTAMP(<TimestampPrecision>) value HS_BeginTime,

b) a TIMESTAMP(<TimestampPrecision>) value HS_EndTime.

6.1.2 HS_History Method

Purpose

Generate a new HS_History value which has the specified TIMESTAMP(<*TimestampPrecision*>) values as the begin time and the end time.

Definition

```
CREATE CONSTRUCTOR METHOD HS_History
    (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
     endOfPeriod TIMESTAMP(<TimestampPrecision>))
    RETURNS HS_History
    SELF AS RESULT
    FOR HS_History
    BEGIN
        -- If the begin time passed to the method is not acceptable,
        --   then raise an exception
        IF beginOfPeriod IS NULL THEN
        -- In case the begin time is null
            SIGNAL SQLSTATE '2FF09' SET MESSAGE_TEXT =
                'beginning of period is a null value';
        ELSEIF endOfPeriod < beginOfPeriod THEN
        -- In case the begin time is later than the end time
            SIGNAL SQLSTATE '2FF10' SET MESSAGE_TEXT =
                'end of period precedes beginning of period';
        ELSEIF endOfPeriod = beginOfPeriod THEN
        -- In case the period has no extent
            SIGNAL SQLSTATE '2FF16' SET MESSAGE_TEXT =
                'empty period';
        ELSEIF HS_GetTransactionTimestamp() <= beginOfPeriod THEN
        -- In case the begin time is equal to or later than the time of
        --   the transaction in which the method is invoked
```

```
            SIGNAL SQLSTATE '2FF23' SET MESSAGE_TEXT =
               'beginning of period succeeds current timestamp';
         ELSEIF HS_GetTransactionTimestamp() <= endOfPeriod THEN
         -- In case the end time is equal to or later than the time of
         --   the transaction in which the method is invoked
            SIGNAL SQLSTATE '2FF24' SET MESSAGE_TEXT =
               'end of period succeeds current timestamp';
         END IF;
         -- Attributes of being constructed HS_History instance are set up
         SET HS_BeginTime = beginOfPeriod;
         SET HS_EndTime = endOfPeriod;
      END
```

Description

1) The *HS_History(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

2) The begin time of the period which this *HS_History* value represents is specified as the input parameter *beginOfPeriod*.

3) The end time of the period which this *HS_History* value represents is specified as the input parameter *endOfPeriod*.

6.1.3 HS_Overlaps Methods

Purpose

Test whether the period of an HS_History value overlaps with the specified period.

Definition

```
   CREATE METHOD HS_Overlaps
      (hs_hist HS_History)
      RETURNS INTEGER
      FOR HS_History
      BEGIN
         -- The value of the parameter hs_hist is
         --   HS_History instance passed to the method.
         --   (It is referred as the term "passed period"
         --    in comments in this method definition.
         --    The begin time and the end time of a passed period are referred
         --    as "passed begin time" and "passed end time", respectively,
         --    in comments in this method definition.)
         --
         -- The instance itself for the method is the HS_History value.
         --   (It is referred as the term "instance period"
         --    in comments in this method definition.)

         DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
         -- the begin time of the instance period
         --   (it is referred as the term "instance begin time"
         --    in comments in this method definition.)
```

```
DECLARE s_et TIMESTAMP(<TimestampPrecision>);
-- the end time of the instance period
--    (it is referred as the term "instance end time"
--     in comments in this method definition.)

DECLARE CurTS TIMESTAMP(<TimestampPrecision>);
-- the timestamp of the transaction
--    in which the method is invoked
--    (it is referred as the term "transaction time"
--     in comments in this method definition.)

SET CurTS = HS_GetTransactionTimestamp();

SET s_bt = SELF.HS_BeginTime;
SET s_et = SELF.HS_EndTime;

IF s_et IS NOT NULL AND hs_hist.HS_EndTime IS NOT NULL THEN
   --
   -- Both of the instance end time and the passed end time
   --    are NOT null.
   --
   RETURN
      CASE
         WHEN hs_hist.HS_BeginTime <= s_bt
                 AND s_bt < hs_hist.HS_EndTime
              OR
              s_bt <= hs_hist.HS_BeginTime
                 AND hs_hist.HS_BeginTime < s_et
            THEN
            --
            -- The instance begin time is within the passed period
            --    or
            --    the passed begin time is within the instance period.
            --    (BETWEEN predicate does not work
            --     because the endpoint of a period is open.)
            1
         ELSE
            0
      END;
ELSEIF s_et IS NULL AND hs_hist.HS_EndTime IS NULL THEN
   --
   -- Both of the instance end time and the passed end time
   --    are null.
   --
   RETURN 1;
ELSEIF s_et IS NULL THEN
   --
   -- Only the instance end time is null
   --
   RETURN
      CASE
         WHEN hs_hist.HS_BeginTime <= s_bt
                 AND s_bt < hs_hist.HS_EndTime
              OR
              s_bt <= hs_hist.HS_BeginTime
                 AND hs_hist.HS_BeginTime < CurTS
```

```
                     THEN
                     --
                     -- The instance begin time is within the passed period
                     --   or
                     --   the passed begin time is within the period
                     --   from the instance begin time to the transaction time.
                     --   (BETWEEN predicate does not work
                     --    because the endpoint of a period is open.)
                     --
                     1
                  ELSE
                     0
               END;
         ELSE
            --
            -- Only the passed end time is null
            --
            RETURN
               CASE
                  WHEN hs_hist.HS_BeginTime <= s_bt AND s_bt < CurTS
                        OR
                        s_bt <= hs_hist.HS_BeginTime
                           AND hs_hist.HS_BeginTime < s_et
                     THEN
                     --
                     -- The instance begin time is within the period
                     --   from the passed begin time to the transaction time
                     --   or
                     --   the passed begin time is within the instance period.
                     --   (BETWEEN predicate does not work
                     --    because the endpoint of a period is open)
                     --
                     1
                  ELSE
                     0
               END;
         END IF;
      END
   CREATE METHOD HS_Overlaps
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS INTEGER
      FOR HS_History
      BEGIN
         RETURN SELF.HS_Overlaps(
                  NEW HS_History(beginOfPeriod, endOfPeriod));
      END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_Overlaps(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_Overlaps(HS_History)* method takes the following input parameter:

a) an HS_History value hs_hist.

6.1.4 HS_Meets Methods

Purpose

Test whether the period of an HS_History value meets the specified period.

Definition

```
CREATE METHOD HS_Meets
   (hs_hist HS_History)
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN
         CASE
            WHEN (SELF.HS_EndTime = hs_hist.HS_BeginTime) THEN
            -- The end time of HS_History instance for the method
            --   is equal to
            --   the begin time of HS_History value passed to the method.
               1
            WHEN (hs_hist.HS_EndTime = SELF.HS_BeginTime) THEN
            -- The end time of HS_History value passed to the method
            --   is equal to
            --   the begin time of HS_History instance for the method.
               1
            ELSE
               0
         END;
   END
CREATE METHOD HS_Meets
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN SELF.HS_Meets(
               NEW HS_History(beginOfPeriod, endOfPeriod));
   END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_Meets(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_Meets(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.5 HS_Precedes Methods

Purpose

Test whether the period of an HS_History value precedes the specified period.

Definition

```
CREATE METHOD HS_Precedes
   (hs_hist HS_History)
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN
         CASE
            WHEN (SELF.HS_EndTime < hs_hist.HS_BeginTime) THEN
            -- The end time of HS_History instance for the method
            --   is earlier than
            --   the begin time of HS_History value passed to the method.
               1
            ELSE
               0
         END;
   END
CREATE METHOD HS_Precedes
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN SELF.HS_Precedes(
               NEW HS_History(beginOfPeriod, endOfPeriod));
   END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_Precedes(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_Precedes(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.6 HS_PrecedesOrMeets Methods

Purpose

Test whether the period of an HS_History value precedes or meets the specified period.

Definition

```
CREATE METHOD HS_PrecedesOrMeets
    (hs_hist HS_History)
    RETURNS INTEGER
    FOR HS_History
    BEGIN
       IF SELF.HS_Precedes(hs_hist) = 1 THEN
       -- The period of HS_History instance for the method
       --   precedes that of HS_History value passed to the method.
          RETURN 1;
       ELSEIF SELF.HS_EndTime = hs_hist.HS_BeginTime THEN
       -- The end time of HS_History instance for the method
       --   is equal to
       --   the begin time of HS_History value passed to the method.
          RETURN 1;
       ELSE
          RETURN 0;
       END IF;
    END
CREATE METHOD HS_PrecedesOrMeets
    (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
     endOfPeriod TIMESTAMP(<TimestampPrecision>))
    RETURNS INTEGER
    FOR HS_History
    BEGIN
       RETURN SELF.HS_PrecedesOrMeets(
                NEW HS_History(beginOfPeriod, endOfPeriod));
    END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_PrecedesOrMeets(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_PrecedesOrMeets(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.7 HS_Succeeds Methods

Purpose

Test whether the period of an HS_History value succeeds the specified period.

Definition

```
CREATE METHOD HS_Succeeds
    (hs_hist HS_History)
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN
         CASE
            WHEN (hs_hist.HS_EndTime < SELF.HS_BeginTime) THEN
            -- The end time of HS_History value passed to the method
            --   is eralier than
            --   the begin time of HS_History instance for the method.
               1
            ELSE
               0
         END;
   END
CREATE METHOD HS_Succeeds
    (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
     endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN SELF.HS_Succeeds(
               NEW HS_History(beginOfPeriod, endOfPeriod));
   END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_Succeeds(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_Succeeds(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.8 HS_SucceedsOrMeets Methods

Purpose

Test whether the period of an HS_History value succeeds or meets the specified period.

Definition

```
CREATE METHOD HS_SucceedsOrMeets
   (hs_hist HS_History)
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      IF SELF.HS_Succeeds(hs_hist) = 1 THEN
      -- The period of HS_History instance for the method
      --   succeeds that of HS_History value passed to the method.
         RETURN 1;
      ELSEIF hs_hist.HS_EndTime = SELF.HS_BeginTime THEN
      -- The end time of HS_History value passed to the method
      --   is equal to
      --   the begin time of HS_History instance for the method.
         RETURN 1;
      ELSE
         RETURN 0;
      END IF;
   END
CREATE METHOD HS_SucceedsOrMeets
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN SELF.HS_SucceedsOrMeets(
                  NEW HS_History(beginOfPeriod, endOfPeriod));
   END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_SucceedsOrMeets(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_SucceedsOrMeets(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.9 HS_Contains Methods

Purpose

Test whether the period of an HS_History value contains the specified TIMESTAMP(<*TimestampPrecision*>) value or the specified period.

Definition

```
CREATE METHOD HS_Contains
    (timePoint TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
      DECLARE s_et TIMESTAMP(<TimestampPrecision>);
      DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

      SET CurTS = HS_GetTransactionTimestamp();

      --
      -- If a given timestamp value indicates a time point in future
      --   after the time of the transaction in which the method is invoked,
      -- then signal an error.
      --
      IF CurTS < timePoint THEN
         SIGNAL SQLSTATE '2FF25' SET MESSAGE_TEXT =
            'a given timestamp value expresses future time';
      END IF;

      SET s_bt = SELF.HS_BeginTime;
      SET s_et = SELF.HS_EndTime;

      RETURN
         CASE
            WHEN (s_bt <= timePoint AND timePoint < s_et) THEN
            -- The time passed to the method is within the period
            --   specified by the HS_History instance for the method.
            --   (BETWEEN predicate does not work
            --    because the endpoint of a period is open)
               1
            WHEN NOT (s_bt <= timePoint AND timePoint < s_et) THEN
            -- The time passed to the method is not within the period
            --   specified by the HS_History instance for the method
            --   (NOT BETWEEN does not work
            --    because the endpoint of a period is open)
               0
            ELSE
               CAST(NULL AS INTEGER)
         END;
   END
CREATE METHOD HS_Contains
    (hs_hist HS_History)
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      -- The value of the parameter hs_hist is
      --   HS_History instance passed to the method.
   --    (It is referred as the term “passed period”
   --     in comments in this method definition.)
   --
   -- The instance itself for the method is the HS_History value.
   --    (It is referred as the term “instance period”
   --     in comments in this method definition.

   DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
   DECLARE s_et TIMESTAMP(<TimestampPrecision>);
   DECLARE p_bt TIMESTAMP(<TimestampPrecision>);
   DECLARE p_et TIMESTAMP(<TimestampPrecision>);
   DECLARE CurTS TIMESTAMP(<TimestampPrecision>);
   -- the timestamp of the transaction
   --    in which the method is invoked
   --    (it is referred as the term “transaction time”
   --     in comments in this method definition.)

   SET CurTS = HS_GetTransactionTimestamp();

   SET s_bt = SELF.HS_BeginTime;
   SET s_et = SELF.HS_EndTime;

   SET p_bt = hs_hist.HS_BeginTime;
   SET p_et = hs_hist.HS_EndTime;

   IF s_et IS NOT NULL THEN
      RETURN
         CASE
            WHEN (s_bt <= p_bt AND p_et <= s_et) THEN
            -- The passed period is within the instance period.
               1
            --
            -- The condition "p_et IS NULL" should be checked
            -- because if the conditions "s_bt <= p_bt" and "p_et IS NULL"
            --  is  both  true,  then  "(s_bt  <=  p_bt  AND  p_et  <=  s_et)"  is
            unknown.
            --
            WHEN NOT (s_bt <= p_bt AND p_et <= s_et) OR p_et IS NULL THEN
            -- The passed period is not within the instance period
            --    or
            --    the passed period has not ended yet.
               0
            ELSE
               CAST(NULL AS INTEGER)
         END;
   ELSE
      RETURN
         CASE
            WHEN (s_bt <= p_bt AND p_et <= CurTS) THEN
            -- The passed period is within the period
            --    from the begin time to the transaction time.
               1
            WHEN NOT (s_bt <= p_bt AND p_et <= CurTS) THEN
               0
            ELSE
               CAST(NULL AS INTEGER)
         END;
   END IF;
END
```

```
CREATE METHOD HS_Contains
   (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
    endOfPeriod TIMESTAMP(<TimestampPrecision>))
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      RETURN SELF.HS_Contains(
               NEW HS_History(beginOfPeriod, endOfPeriod));
   END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_Contains(TIMESTAMP(<TimestampPrecision>))* method takes the following input parameter:

   a) a TIMESTAMP(<TimestampPrecision>) value timePoint.

3) The *HS_Contains(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

4) The *HS_Contains(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.10 HS_Equals Methods

Purpose

Test whether the period of an HS_History value is equal to the specified period.

Definition

```
CREATE METHOD HS_Equals
   (hs_hist HS_History)
   RETURNS INTEGER
   FOR HS_History
   BEGIN
      -- The value of the parameter hs_hist is
      --   HS_History instance passed to the method.
      --   (It is referred as the term "passed period"
      --    in comments in this method definition.)
      --
      -- The instance itself for the method is the HS_History value.
      --   (It is referred as the term "instance period"
      --    in comments in this method definition.

      DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
      DECLARE s_et TIMESTAMP(<TimestampPrecision>);
      DECLARE p_bt TIMESTAMP(<TimestampPrecision>);
      DECLARE p_et TIMESTAMP(<TimestampPrecision>);
```

```
         SET s_bt = SELF.HS_BeginTime;
         SET s_et = SELF.HS_EndTime;
         SET p_bt = hs_hist.HS_BeginTime;
         SET p_et = hs_hist.HS_EndTime;

         RETURN
            CASE
               WHEN (s_bt = p_bt AND s_et = p_et) THEN
               -- The begin time and end time of the instance period
               --   coincide with those of the passed period, respectively.
                  1
               WHEN NOT (s_bt = p_bt AND s_et = p_et) THEN
               -- The begin time and end time of the instance period
               --   do not coincide with those of the passed period,
               respectively.
                  0
               ELSE
                  CAST(NULL AS INTEGER)
            END;
      END
   CREATE METHOD HS_Equals
      (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
       endOfPeriod TIMESTAMP(<TimestampPrecision>))
      RETURNS INTEGER
      FOR HS_History
      BEGIN
         RETURN SELF.HS_Equals(
                  NEW HS_History(beginOfPeriod, endOfPeriod));
      END
```

Description

1) The return value of methods defined in this subclause is 1 (one), 0 (zero) or null value. The return value 1 (one) means that the result of the evaluation of predicate condition is True. The return value 0 (zero) means that the result of the evaluation of predicate condition is False. The null return value means that the result of the evaluation of predicate condition is Unknown.

2) The *HS_Equals(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

3) The *HS_Equals(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.11 HS_MonthInterval Method

Purpose

Obtain the length of the period of an HS_History value as a year-month interval.

Definition

```
   CREATE METHOD HS_MonthInterval()
```

```
      RETURNS INTERVAL YEAR TO MONTH
      FOR HS_History
      BEGIN
         DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
         -- the begin time of HS_History instance itself for the method
         DECLARE s_et TIMESTAMP(<TimestampPrecision>);
         -- the end time of HS_History instance itself for the method

         SET s_bt = SELF.HS_BeginTime;
         SET s_et = SELF.HS_EndTime;

         -- In case that a period has not ended yet,
         --   the value of the interval is unknown.
         IF s_et IS NULL THEN
            RETURN CAST(NULL AS INTEGER);
         END IF;

         -- The year-month interval value of HS_History instance is calculated.
         RETURN (s_et - s_bt) YEAR TO MONTH;
      END
```

Description

1) The *HS_MonthInterval()* method has no input parameters.

6.1.12 HS_DayInterval Method

Purpose

Obtain the length of the period of an HS_History value as a day-time interval.

Definition

```
   CREATE METHOD HS_DayInterval()
      RETURNS INTERVAL DAY TO SECOND
      FOR HS_History
      BEGIN
         DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
         -- the begin time of HS_History instance itself for the method
         DECLARE s_et TIMESTAMP(<TimestampPrecision>);
         -- the end time of HS_History instance itself for the method

         SET s_bt = SELF.HS_BeginTime;
         SET s_et = SELF.HS_EndTime;

         -- In case that a period has not ended yet,
         --   the value of the interval is unknown.
         IF s_et IS NULL THEN
            RETURN CAST(NULL AS INTEGER);
         END IF;

         -- The day-second interval value of HS_History instance is calculated.
         RETURN (s_et - s_bt) DAY TO SECOND;
      END
```

Description

1) The *HS_DayInterval()* method has no input parameters.

6.1.13 HS_Intersect Methods

Purpose

Generate a new HS_History value with the period which is the overlap of the period of an HS_History value and the specified period.

Definition

```
CREATE METHOD HS_Intersect
   (hs_hist HS_History)
   RETURNS HS_History
   FOR HS_History
   BEGIN
      -- The value of the parameter hs_hist is
      --   HS_History instance passed to the method.
      --   (It is referred as the term "passed period"
      --    in comments in this method definition.)
      --
      -- The instance itself for the method is the HS_History value.
      --   (It is referred as the term "instance period"
      --    in comments in this method definition.

      DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
      -- the begin time of the instance period
      --   (it is referred as the term "instance begin time"
      --    in comments in this method definition.)

      DECLARE s_et TIMESTAMP(<TimestampPrecision>);
      -- the end time of the instance period
      --   (it is referred as the term "instance end time"
      --    in comments in this method definition.)

      DECLARE p_bt TIMESTAMP(<TimestampPrecision>);
      -- the begin time of the passed period
      --   (it is referred as the term "passed begin time"
      --    in comments in this method definition.)

      DECLARE p_et TIMESTAMP(<TimestampPrecision>);
      -- the end time of the passed period
      --   (it is referred as the term "passed end time"
      --    in comments in this method definition.)

      DECLARE st TIMESTAMP(<TimestampPrecision>);
      DECLARE et TIMESTAMP(<TimestampPrecision>);

      DECLARE resultHSHist HS_History;

      SET s_bt = SELF.HS_BeginTime;
      SET s_et = SELF.HS_EndTime;

      SET p_bt = hs_hist.HS_BeginTime;
      SET p_et = hs_hist.HS_EndTime;
```

```
    -- The maximum value
    --   of the passed begin time and the instance begin time
    --   is obtained.
    -- This is the begin time of a returned HS_History value.

    SET st = CASE
                WHEN p_bt < s_bt THEN s_bt
                ELSE p_bt
             END;

    -- Unless either of the passed end time or the instance end time is
    null,
    --   then obtain the minimum value
    --   of the passed begin time and the instance begin time.
    -- If either of the passed end time or the instance end time is null
    --   then the returned end time is null, tentatively.
    SET et = CASE
                WHEN s_et < p_et THEN s_et
                WHEN p_et <= s_et THEN p_et
                ELSE NULL
             END;

    -- Hereafter, the returned end time is determined and
    --   the retuned HS_History value is established.
    IF s_et IS NULL AND p_et IS NULL THEN
    -- If both of the passed end time or the instance end time is null
    --   then the returned end time is null.
       RETURN NEW HS_History(st, CAST(NULL AS
       TIMESTAMP(<TimestampPrecision>)));
    ELSEIF st < et THEN
    -- If the returned begin time is earlier than the returned end time
    --   (this condition implies the returned end time is not tentative
    null),
    --   then the returned end time is the above-obtained minimum end time.
       SET resultHSHist = NEW HS_History(
             st, et);
       RETURN resultHSHist;
    ELSEIF et IS NULL THEN
       --
       -- If just one of s_et(instance end time) and p_et(passed end time)
       --   is NULL value
       --   then the returned end time is the non-null one of those.
       -- If both of those is null, then the returned end time is null.
       --
       SET resultHSHist = NEW HS_History(
             st, COALESCE(s_et, p_et));
       RETURN resultHSHist;
    ELSE
    -- IF the returned HS_Histor value has no extent,
    --   then raise an exception.
       SIGNAL SQLSTATE '2FF19' SET MESSAGE_TEXT =
          'result of the intersect operation is empty period';
    END IF;
 END
```

```
CREATE METHOD HS_Intersect
    (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
     endOfPeriod TIMESTAMP(<TimestampPrecision>))
    RETURNS HS_History
    FOR HS_History
    BEGIN
       RETURN SELF.HS_Intersect(
                NEW HS_History(beginOfPeriod, endOfPeriod));
    END
```

Description

1) The *HS_Intersect(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

2) The *HS_Intersect(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.14 HS_Union Methods

Purpose

Generate a new HS_History value with the period which is the union of the period of an HS_History value and the specified period.

Definition

```
CREATE METHOD HS_Union
    (hs_hist HS_History)
    RETURNS HS_History
    FOR HS_History
    BEGIN
       -- The value of the parameter hs_hist is
       --   HS_History instance passed to the method.
       --   (It is referred as the term "passed period"
       --    in comments in this method definition.)
       --
       -- The instance itself for the method is the HS_History value.
       --   (It is referred as the term "instance period"
       --    in comments in this method definition.

       IF SELF.HS_EndTime < hs_hist.HS_BeginTime
             OR hs_hist.HS_EndTime < SELF.HS_BeginTime THEN
       -- The instance period and the passed period are non-contiguous and
       disjoint.
          SIGNAL SQLSTATE '2FF20' SET MESSAGE_TEXT =
             'the two periods do not meet or overlap';
       ELSE
          BEGIN
             DECLARE BeginTime TIMESTAMP(<TimestampPrecision>);
             DECLARE EndTime TIMESTAMP(<TimestampPrecision>);
             DECLARE resultHSHist HS_History;
             -- The returned begin time is the minimum of
```

```
                --   begin times of the instance period and the passed period.
                -- If either of end times of the instance period or the passed
                period
                --   is null, then the returned end time is null.
                -- If neither of end times of
                --   the instance period nor the passed period is null,
                --   then the returned end time is the maximum of those.
                SELECT MIN(BTime),
                      CASE WHEN(EVERY(ETime IS NOT NULL)) THEN MAX(ETime)
                         ELSE NULL END
                   INTO BeginTime, EndTime
                   FROM VALUES
                      (SELF.HS_BeginTime, SELF.HS_EndTime),
                      (hs_hist.HS_BeginTime, hs_hist.HS_EndTime)
                         AS Bndy(BTime, ETime);
                SET resultHSHist = NEW HS_History(
                      BeginTime, EndTime);
                RETURN resultHSHist;
             END;
          END IF;
       END
    CREATE METHOD HS_Union
       (beginOfPeriod TIMESTAMP(<TimestampPrecision>),
        endOfPeriod TIMESTAMP(<TimestampPrecision>))
       RETURNS HS_History
       FOR HS_History
       BEGIN
          RETURN SELF.HS_Union(
                   NEW HS_History(beginOfPeriod, endOfPeriod));
       END
```

Description

1) The *HS_Union(TIMESTAMP(<TimestampPrecision>), TIMESTAMP(<TimestampPrecision>))* method takes the following input parameters:

   a) a TIMESTAMP(<TimestampPrecision>) value beginOfPeriod,

   b) a TIMESTAMP(<TimestampPrecision>) value endOfPeriod.

2) The *HS_Union(HS_History)* method takes the following input parameter:

   a) an HS_History value hs_hist.

6.1.15 HS_Except Methods

Purpose

Generate a new HS_History value with the period obtained from the period of an HS_History value except for the specified period.

Definition

```
CREATE METHOD HS_Except
   (hs_hist HS_History)
   RETURNS HS_History
   FOR HS_History
   BEGIN
      -- The value of the parameter hs_hist is
      --   HS_History instance passed to the method.
      --   (It is referred as the term "passed period"
      --    in comments in this method definition.
      --    The begin time and the end time of a passed period are referred
      --    as "passed begin time" and "passed end time", respectively,
      --    in comments in this method definition.)
      --
      -- The instance itself for the method is the HS_History value.
      --   (It is referred as the term "instance period"
      --    in comments in this method definition.
      --    The begin time and the end time of the instance period are
      referred
      --    as "instance begin time" and "instance end time"
      --    in comments in this method definition.)

      DECLARE resultHSHist HS_History;
      IF SELF.HS_EndTime <= hs_hist.HS_BeginTime
            OR hs_hist.HS_EndTime <= SELF.HS_BeginTime THEN
      -- The instance period and the passed period are disjoint.
      -- This condition is classified into two.
      -- The first is that the instance end time is equal to or earlier than
      --   the passed begin time.(This implies the instance end time is not
      null.)
      -- The second is that the passed end time is equal to or earlier than
      --   the instance begin time.(This implies the passed end time is not
      null.)
         RETURN SELF; -- The returned period is the instance period itself.
      ELSEIF SELF.HS_BeginTime < hs_hist.HS_BeginTime
            AND (SELF.HS_EndTime <= hs_hist.HS_EndTime
               OR hs_hist.HS_EndTime IS NULL) THEN
      -- The instance period and the passed period intersect and
      --   the instance period begins earlier than the passed period
      --   and ends at the same time or earlier than the passed period.
         SET resultHSHist = NEW HS_History(
               SELF.HS_BeginTime, hs_hist.HS_BeginTime);
      -- The returned period is the part of the instance period
      --    that precedes the passed period.
         RETURN resultHSHist;
      ELSEIF hs_hist.HS_BeginTime <= SELF.HS_BeginTime
            AND (hs_hist.HS_EndTime < SELF.HS_EndTime
            OR (SELF.HS_EndTime IS NULL
               AND hs_hist.HS_EndTime IS NOT NULL)) THEN
      -- The instance period and the passed period intersect and
      --   the instance period begins at the same time
      --   or later than the passed period
      --   and ends later than the instance period.
         SET resultHSHist = NEW HS_History(
               hs_hist.HS_EndTime, SELF.HS_EndTime);
      -- The returned period is the part of the instance period
      --    that succeeds the passed period.
         RETURN resultHSHist;
      ELSEIF SELF.HS_BeginTime < hs_hist.HS_BeginTime
```