

<b>Final Committee Draft ISO/IEC FCD2 13249-7</b>	
Date: <b>2009-12-30</b>	Reference number: ISO/JTC 1/SC 32 <b>N1941</b>
Supersedes document SC 32N1827	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange  Secretariat: USA (ANSI)	<p>Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by:</p> <p style="text-align: center;"><b>2010-04-30</b></p> <p>Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.</p>
---	--

<p>ISO/IEC: FCD 13249-7:2009(E)</p> <p>Title: Information technology - SQL Multimedia and Application Packages - Part 7: History</p> <p>Project: 1.32.04.03.07.00</p>
---

<p>Introductory note: Text for FCD2 13249-7; see FCD disposition of comments in N1942; this text is sent to NBs for 4 month letter ballot. The ballot starts 2009-12-30.</p> <p>Medium: E</p> <p>No. of pages: 122</p>
--

Dr. Timothy Schoechle, Secretary, ISO/IEC JTC 1/SC 32  
Farance Inc \*, 3066 Sixth Street, Boulder, CO, United States of America  
Telephone: +1 303-443-5490; E-mail: Timothy@Schoechle.org  
available from the JTC 1/SC 32 WebSite <http://www.jtc1sc32.org/>  
\*Farance Inc. administers the ISO/IEC JTC 1/SC 32 Secretariat on behalf of ANSI

*Final Committee Draft*  
**2nd**

## **Information technology — Database languages — SQL Multimedia and Application Packages — Part 7: History**

*Technologies de l'information – Langues de bases de données — Multimédia SQL et paquetages d'application — Partie 7: Histoire*

### **Warning**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International Standard  
Document subtype:  
Document stage: (30) Committee  
Document language: E



blank page

# Contents

	Page
Foreword.....	vi
Introduction.....	vii
1 Scope.....	1
2 Normative references.....	2
2.1 ISO/IEC JTC 1 standards.....	2
3 Terms and definitions, notations, and conventions.....	3
3.1 Definitions and concepts.....	3
3.1.1 Definitions taken from ISO/IEC 9075-1.....	3
3.1.2 Definitions taken from ISO/IEC 9075-2.....	3
3.1.3 Concepts taken from ISO/IEC 9075-1.....	3
3.1.4 Concepts taken from ISO/IEC 9075-2.....	4
3.1.5 Syntactic elements taken from ISO/IEC 9075-2.....	4
3.1.6 Definitions provided in Part 1.....	5
3.1.7 Definitions provided in Part 7.....	5
3.2 Notations.....	6
3.2.1 Notations provided in Part 1.....	6
3.2.2 Notations provided in Part 7.....	6
3.3 Conventions.....	7
4 Concepts.....	8
4.1 Overview.....	8
4.1.1 Tracked Table and History Table.....	8
4.1.2 Concept of Transaction Timestamp.....	8
4.1.3 Operations of Tracked Table.....	9
4.1.4 Operations on time periods.....	9
4.1.5 Concept of Period Normalization.....	11
4.2 Structure of History Table.....	12
4.3 Generating History Table and Storing History Row to History Table.....	13
4.4 Retrieving a History Table.....	13
4.5 Types representing history rows.....	14
4.5.1 HS_History type.....	14
4.5.2 <TableTypeIdentifier> type.....	17
4.6 Complementary SQL-invoked regular functions.....	18
4.6.1 Constructor method of the HS_History type.....	18
4.6.2 Methods of the HS_History type for treating a period.....	18
4.6.3 Methods of the <TableTypeIdentifier> type.....	19
4.7 The History Information Schema.....	20
5 History Procedures.....	21
5.1 HS_CreateHistory Procedure and Sub Procedures.....	21
5.1.1 HS_CreateHistory Procedure.....	21
5.1.2 HS_CreateHistoryErrorCheck Procedure.....	22
5.1.3 HS_CreateHistoryTableType Procedure.....	24
5.1.4 HS_CreateHistoryTable Procedure.....	25
5.1.5 HS_CreateUpdateTrigger Procedure.....	26
5.1.6 HS_CreateInsertTrigger Procedure.....	28
5.1.7 HS_CreateDeleteTrigger Procedure.....	30
5.1.8 HS_CreateHistoryTableMethod Procedure.....	31
5.1.9 HS_CreatePNormalizeMethod Procedure.....	32
5.1.10 HS_InitializeHistoryTable Procedure.....	36
5.2 HS_DropHistory Procedure and Sub Procedures.....	37

5.2.1	HS_DropHistory Procedure .....	37
5.2.2	HS_DropHistoryErrorCheck Procedure .....	38
5.2.3	HS_DropHistoryTableTypeMethod Procedure.....	39
5.2.4	HS_DropHistoryTrigger Procedure .....	40
5.2.5	HS_DropHistoryTable Procedure .....	42
5.2.6	HS_DropHistoryTableType Procedure .....	43
5.3	Utility Procedures for History .....	44
5.3.1	HS_CreateCommaSeparatedPrimaryKeyList Procedure.....	44
5.3.2	HS_CreateCommaSeparatedPrimaryKeyAndTypeList Procedure.....	46
5.3.3	HS_CreatePrimaryKeySelfJoinCondition Procedure.....	48
5.3.4	Functions for extracting an identifier.....	50
5.3.5	HS_CreateCommaSeparatedTrackedColumnList Procedure.....	53
5.3.6	HS_CreateCommaSeparatedTrackedColumnAndTypeList Procedure.....	55
5.3.7	Functions for constructing an identifier and <IdentifierLength> .....	57
5.3.8	HS_GetPrimaryKeys function.....	60
5.3.9	HS_GetTransactionTimestamp function .....	61
5.4	<TableNameLength> and <ColumnNameLength> .....	62
5.5	Schema for <TableTypeIdentifier> Type .....	63
5.6	<TimestampPrecision> .....	64
6	History Types .....	65
6.1	HS_History Type and Routines .....	65
6.1.1	HS_History Type .....	65
6.1.2	HS_History Method .....	69
6.1.3	HS_Overlaps Methods .....	70
6.1.4	HS_Meets Methods .....	72
6.1.5	HS_Precedes Methods.....	73
6.1.6	HS_PrecedesOrMeets Methods.....	74
6.1.7	HS_Succeeds Methods .....	75
6.1.8	HS_SucceedsOrMeets Methods .....	76
6.1.9	HS_Contains Methods .....	78
6.1.10	HS_Equals Methods.....	80
6.1.11	HS_MonthInterval Method .....	81
6.1.12	HS_DayInterval Method .....	82
6.1.13	HS_Intersect Methods.....	83
6.1.14	HS_Union Methods .....	85
6.1.15	HS_Except Methods.....	86
6.2	<TableTypeIdentifier> Type and Routines .....	88
6.2.1	<TableTypeIdentifier> Type .....	88
6.2.2	HS_HistoryTable Method .....	89
6.2.3	HS_PNormalize Methods .....	90
7	SQL/MM History Information Schema.....	94
7.1	Introduction .....	94
7.2	HS_TRACKED_TABLES view .....	95
7.3	HS_TRACKED_COLUMNS view.....	96
8	SQL/MM History Definition Schema .....	97
8.1	Introduction.....	97
8.2	HS_TRACKED_TABLES base table.....	98
8.3	HS_TRACKED_COLUMNS base table.....	99
9	Status Codes .....	100
10	Conformance.....	102
10.1	Requirements for conformance.....	102
10.2	Features of ISO/IEC 9075 required in this part of ISO/IEC 13249.....	102
10.3	Claims of conformance.....	102
Annex A	.....	104
A.1	Introduction .....	104
A.2	Storing History Rows.....	104
A.3	Example of Queries to History Table.....	108

**Bibliography.....112**

## 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 3.

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.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 13249 may be the subject of patent rights. ISO and IEC 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*

Annex A and the Bibliography of this part of ISO/IEC 13249 are for information only.

## Introduction

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

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

The organization of this part of ISO/IEC 13249 is as follows:

- 1) Clause 1, "Scope", specifies the scope of this part of ISO/IEC 13249.
- 2) Clause 2, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 13249, constitute provisions of this part of ISO/IEC 13249.
- 3) Clause 3, "Terms and definitions, notations, and conventions", defines the notations and conventions used in this part of ISO/IEC 13249.
- 4) Clause 4, "Concepts", presents concepts used in the definition of this part of ISO/IEC 13249.
- 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 part of ISO/IEC 13249.
- 10) Clause 10, "Conformance", defines the criteria for conformance to this part of ISO/IEC 13249.

In the text of this part of ISO/IEC 13249, Clauses begin a new page. Any resulting blank space is not significant.



# Information technology — Database languages — SQL Multimedia and Application Packages — Part 7: History

## 1 Scope

ISO/IEC 13249 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 part of ISO/IEC 13249:

- a) introduces the History part of ISO/IEC 13249,
- b) gives the references necessary for this part of ISO/IEC 13249,
- c) defines notations and conventions specific to this part of ISO/IEC 13249,
- d) defines concepts specific to this part of ISO/IEC 13249,
- e) defines history user-defined types and their associated routines.

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

- 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.
- History user-defined types and routines provide the means to query the recorded changes for such a table.
- 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 part is limited to support for history when there is no changes to the definition of the tracked columns of a tracked table. The following operations are not supported in this standard.

- DROP COLUMN operation to a tracked column of a tracked table.
- ALTER COLUMN operation to a tracked column of a tracked table.

The scope of this part is limited to support for history when a tracked table has primary key columns.

## 2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this part of ISO/IEC 13249. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 13249 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of IEC and ISO maintain registers of currently valid International Standards.

### 2.1 ISO/IEC JTC 1 standards

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, notations, and conventions

#### 3.1 Definitions and concepts

For the purpose of this part of ISO/IEC 13249, the following definitions and concepts apply.

##### 3.1.1 Definitions taken from ISO/IEC 9075-1

For the purposes of this document, the following definitions defined in ISO/IEC 9075-1 apply.

- 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 Definitions taken from ISO/IEC 9075-2

For the purposes of this document, the following definitions defined in ISO/IEC 9075-2 apply.

- a) distinct (of a pair of comparable values)
- b) equal (of a pair of comparable values)
- c) identical (of a pair of values)
- d) SQL parameter
- e) SQL routine
- f) structured type
- g) variable-length

##### 3.1.3 Concepts taken from ISO/IEC 9075-1

For the purposes of this document, the following concepts defined in ISO/IEC 9075-1 are used.

- 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

### **3.1.4 Concepts taken from ISO/IEC 9075-2**

For the purposes of this document, the following concepts defined in ISO/IEC 9075-2 are used.

- a) applicable role
- b) authorization identifier
- c) default unqualified schema name
- d) default catalog name
- e) enabled authorization identifier
- f) parameter
- g) procedure
- h) SQL-path
- i) SQL-session context

### **3.1.5 Syntactic elements taken from ISO/IEC 9075-2**

For the purposes of this document, the following syntactic elements(BNF nonterminal symbols) defined in ISO/IEC 9075-2 are used.

- a) <catalog name>
- b) <column name>
- c) <comma>
- d) <data type or domain name>
- e) <delimited identifier body>
- f) <delimited identifier>
- g) <double quote>
- h) <equals operator>
- i) <identifier body>
- j) <local or schema qualifier>
- k) <qualified identifier>

- l) <quote>
- m) <regular identifier>
- n) <rollback statement>
- o) <schema name list>
- p) <schema name>
- q) <space>
- r) <table name>
- s) <Unicode delimiter body>
- t) <Unicode delimiter identifier>
- u) <unqualified schema name>

### 3.1.6 Definitions provided in Part 1

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

### 3.1.7 Definitions provided in Part 7

For the purposes of this document, the following terms and definitions apply.

#### 3.1.5.1

##### **contiguous periods**

a sequence of two or more periods, such that, for all  $1 < i \leq n$  (where  $n$  is the number of periods), the begin time of the  $i$ -th period is equal to the end time of the  $(i-1)$ -th period.

#### 3.1.5.2

##### **history row**

a row in a history table

#### 3.1.5.3

##### **history row set**

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

#### 3.1.5.4

##### **history table**

a 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

#### 3.1.5.5

##### **period**

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

NOTE 1 In this standard a period includes the begin time but not the end time.

#### 3.1.5.6

##### **period normalization**

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

**3.1.5.7**

**period-normalized table**

the table resulting from selecting one or more of the columns in a history table, including the column(s) corresponding to the primary key of the tracked table, and applying period normalization to rows that are otherwise identical. 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

**3.1.5.8**

**tracked column**

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

NOTE 2 the tracked columns of a tracked table must include the primary key of that table.

**3.1.5.9**

**tracked row**

a row in a tracked table

**3.1.5.10**

**tracked table**

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

**3.1.5.11**

**transaction timestamp**

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

NOTE 3 this value is implementation-dependent, preferably corresponding to the end of an SQL-transaction

**3.2 Notations**

**3.2.1 Notations provided in Part 1**

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

**3.2.2 Notations provided in Part 7**

This part of ISO/IEC 13249 uses the prefix 'HS\_' for views, base tables, user-defined types, attributes and SQL-invoked routine names.

This part of ISO/IEC 13249 uses the following representation in a figure for a table that includes the column of HS\_Hist of the structured type, HS\_History.

<i>&lt;column name&gt; of Predefined Type</i>	<i>&lt;column name&gt; of Predefined Type</i>	<i>...</i>	<i>HS_Hist (HS_BeginTime, HS_EndTime)</i>
<i>Column Value</i>	<i>Column Value</i>	<i>...</i>	<i>(Attribute Value, Attribute Value)</i>
<i>Column Value</i>	<i>Column Value</i>	<i>...</i>	<i>(Attribute Value, Attribute Value)</i>
<i>...</i>	<i>...</i>	<i>...</i>	<i>...</i>

### 3.3 Conventions

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

## 4 Concepts

### 4.1 Overview

This part of ISO/IEC 13249 provides user-defined types and routines that enable a user to specify that, for selected columns of a table, inserts, deletes and updates of those columns are tracked such that all changes to those columns are recorded, and subsequently to query those changes.

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 columns corresponding to all tracked columns of the tracked table and a column of a structured type for a period of a history row.

Example:

Tracked Table TT

ID	Column A	Column B	Column C
----	----------	----------	----------

History Table of Tracked Table TT

ID	Column A	Column B	HS_Hist (HS_BeginTime, HS_EndTime)
----	----------	----------	------------------------------------

In this example, ID is a primary key column of the tracked table TT. Column A and Column B are tracked columns of the tracked table TT.

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 to the tracked table TT.

A row in the history table, namely history row, represents the values of the columns (in this example, ID, Column A, and Column B) that existed from begin time to end time. Once a begin/end time is set to a certain non-NULL timestamp value, the value is not changed any more.

#### 4.1.2 Concept of Transaction Timestamp

When an insert operation, an update operation or a delete operation is executed, the value of a begin time and an end time of a inserted history row are set to the value which is based on CURRENT\_TIMESTAMP. But if multiple DML operations are executed in the same SQL-transaction, these operations typically see different values of CURRENT\_TIMESTAMP. This is an especially serious problem if more than one table is being tracked.

Transaction timestamp is a timestamp which satisfies the following requirements:

- The value of a transaction timestamp is the same in an SQL-transaction.

- The value of transaction timestamp is implementation-dependent and can be the value of any CURRENT\_TIMESTAMP that exists from the start of the SQL-transaction until the completion of the 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 of 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 of Tracked Table

A history table of a tracked table is created automatically by invoking the HS\_CreateHistory procedure provided in this part of ISO/IEC 13249.

When the HS\_CreateHistory procedure is invoked, all rows in the tracked table is inserted into the history table, and the value of begin time of all history rows is set to the value of transaction timestamp.

When an insert operation for 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.

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

- 1) in the history row set for that row, the row with the latest begin time has its end time set to the transaction timestamp, and
- 2) a history row recording the tracked columns for that row is inserted into the history table with its begin time set to the transaction timestamp.

When a delete operation for a tracked table is executed, for each row that is deleted, in the history row set for that row the row with the latest begin time has its end time set to the transaction timestamp.

#### 4.1.4 Operations on time periods

In order to query a history table, an application must be able to specify conditions for the begin time or the end time that it is interested in. Also an application must be able 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 the begin of period is inclusive and the right parentheses shows 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  $b1 \leq b2$  and  $e2 \leq e1$  are both true.
  - false if and only if either of  $b1 \leq b2$  or  $e2 \leq e1$  is false.
- $X$  Meets  $Y$  is:
  - true if and only if  $e1 = b2$  is true.

- false if and only if  $e1 = b2$  is false.

- X Precedes Y is:

- true if and only if  $e1 < b2$  is true.

- false if and only if  $e1 < b2$  is false.

- X Equals Y is:

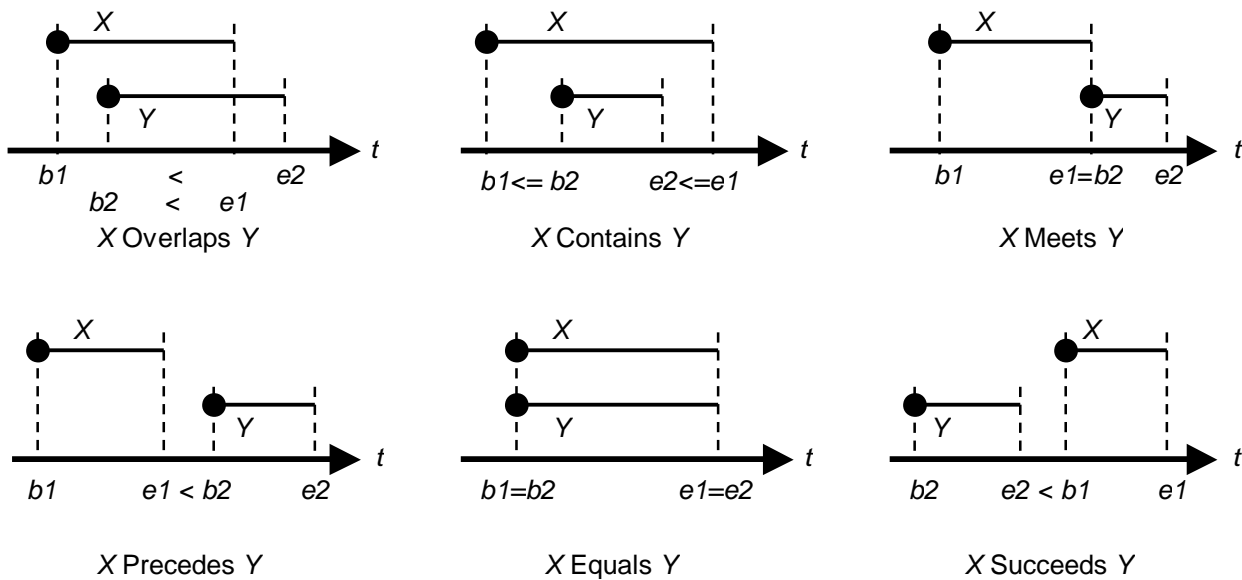
- true if and only if  $b1 = b2$  and  $e1 = e2$  are both true.

- 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.



In this part of ISO/IEC 13249, 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.

In the sub-section, we consider the case where a history table has two tracked columns. 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 wants to use a history table on only one of the two tracked columns that were originally selected. Since it was possible that tracked rows were added to the originally specified history table for changes in the other column the application would have to specify a complicated query to determine the actual length of the periods when the values of the selected column were not changed.

The operation of “period normalization” is introduced in this part of ISO/IEC 13249 to permit 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 makes 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 identical across a set of history rows which have contiguous periods.

In each such set of history rows, the values of the selected columns are identical from the begin time of the oldest history row in the set to the end time of the latest history row in the set. Therefore, the period-normalized history table holds one row for each set of history rows in each history row set which have the following values:

- 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.

Example

History Table of Tracked Table TT

ID	Column A	Column B	HS_Hist (HS_BeginTime, HS_EndTime)
1	A1	B1	(T1, T2)
1	A1	B2	(T2, T3)
1	A2	B2	(T3, T4)
1	A2	B3	(T4, T5)
1	A3	B3	(T5, T6)
1	A4	B3	(T6, T7)
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:

ID	Column A	Column B	HS_Hist (HS_BeginTime, HS_EndTime)
1	NULL	B1	(T1, T2)
1	NULL	B2	(T2, T4)
1	NULL	B3	(T4, T7)
1	NULL	B4	(T7, NULL)

#### 4.2 Structure of History Table

Whenever an update/insert operation occurs on a tracked row, a row is inserted into a history table. 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 adjustments must be maintained 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 must be 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 must be 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:

- attributes correspond to all tracked columns of the tracked table;
- HS\_Hist: an HS\_History value to store the period which is associated with a history row;

### 4.3 Generating History Table and Storing History Row to History Table

The HS\_CreateHistory procedure is provided to generate 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 defined:

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

These triggers are automatically generated 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 unchanged. The HS\_PNormalize method returns a table that is the result of period normalization of selected 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:

- a) *V*.HS\_BeginTime cannot be null.
- b) *V*.HS\_BeginTime is less than CURRENT\_TIMESTAMP.
- 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 strictly greater than *V*.HS\_BeginTime but less than or equal to CURRENT\_TIMESTAMP. 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 specified period;
- HS\_Overlaps(*hs\_hist* HS\_History)  
RETURNS INTEGER:  
Test whether the period of an HS\_History value overlaps with the specified period;
- HS\_Meets(*timePoint* TIMESTAMP(<TimestampPrecision>))  
RETURNS INTEGER:  
Test whether the period of an HS\_History value meets the specified period;
- HS\_Meets(*hs\_hist* HS\_History)  
RETURNS INTEGER:  
Test whether the period of an HS\_History value meets the specified period;

- HS\_Precedes(timePoint TIMESTAMP(<TimestampPrecision>))  
RETURNS INTEGER:  
Test whether the period of an HS\_History value precedes the specified period;
- HS\_Precedes(hs\_hist HS\_History)  
RETURNS INTEGER:  
Test whether the period of an HS\_History value precedes the specified period;
- HS\_PrecedesOrMeets(timePoint TIMESTAMP(<TimestampPrecision>))  
RETURNS INTEGER:  
Test whether the period of an HS\_History value precedes or meets the specified period;
- 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(timePoint TIMESTAMP(<TimestampPrecision>))  
RETURNS INTEGER:  
Test whether the period of an HS\_History value succeeds the specified period;
- HS\_Succeeds(hs\_hist HS\_History)  
RETURNS INTEGER:  
Test whether the period of an HS\_History value succeeds the specified period;
- HS\_SucceedsOrMeets(timePoint TIMESTAMP(<TimestampPrecision>))  
RETURNS INTEGER:  
Test whether the period of an HS\_History value succeeds or is met by the specified period;
- HS\_SucceedsOrMeets(hs\_hist HS\_History)  
RETURNS INTEGER:  
Test whether the period of an HS\_History value succeeds or is met by 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 specified period;
- 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 specified period;
- 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 year-month interval;

- HS\_DayInterval()  
RETURNS INTERVAL DAY TO SECOND:  
Obtain the length of the period of an HS\_History value as day-time interval;
  
- HS\_Intersect  
(beginOfPeriod TIMESTAMP(<TimestampPrecision>),  
endOfPeriod TIMESTAMP(<TimestampPrecision>))  
RETURNS HS\_History:  
Generate a new HS\_History value with period which is an overlap of the period of an HS\_History value and the specified period;
  
- HS\_Intersect(hs\_hist HS\_History)  
RETURNS HS\_History:  
Generate a new HS\_History value with period which is an 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 period which is the union of the period of an HS\_History value and the specified period;
  
- HS\_Union(hs\_hist HS\_History)  
RETURNS HS\_History:  
Generate a new HS\_History value with 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 period obtained from the period of an HS\_History value except for the specified period;
  
- HS\_Except(hs\_hist HS\_History)  
RETURNS HS\_History:  
Generate a new HS\_History value with 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:

- attributes correspond to all tracked columns of a tracked table;
- HS\_Hist: a HS\_History value;

### 4.5.2.2 Methods of the <TableTypeIdentifier> type

The type <TableTypeIdentifier> provides the following methods for public use:

- HS\_HistoryTable()  
RETURNS TABLE(<AllPrimaryKeyColumnsDef>, <AllTrackedColumnsDef>, HS\_Hist HS\_History):  
Obtain whole history table;
- HS\_PNormalize(targetColumn CHARACTER VARYING(<ColumnNameLength>))  
RETURNS TABLE(<AllPrimaryKeyColumnsDef>, <AllTrackedColumnsDef>, HS\_Hist HS\_History):  
Obtain a period-normalized table of the specified column of a history table;
- HS\_PNormalize(targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)  
RETURNS TABLE(<AllPrimaryKeyColumnsDef>, <AllTrackedColumnsDef>, HS\_Hist HS\_History):  
Obtain a period-normalized table of the specified columns of a history table;

#### 4.6 Complementary SQL-invoked regular functions

To ease conformance for implementation of this part of ISO/IEC 13249, 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>)	HS_Meets_T
HS_History	HS_Meets	HS_History	HS_Meets_H
HS_History	HS_Precedes	TIMESTAMP(<TimestampPrecision>)	HS_Precedes_T
HS_History	HS_Precedes	HS_History	HS_Precedes_H
HS_History	HS_PrecedesOrMeets	TIMESTAMP(<TimestampPrecision>)	HS_PrecedesOrMeets_T
HS_History	HS_PrecedesOrMeets	HS_History	HS_PrecedesOrMeets_H
HS_History	HS_Succeeds	TIMESTAMP(<TimestampPrecision>)	HS_Succeeds_T
HS_History	HS_Succeeds	HS_History	HS_Succeeds_H
HS_History	HS_SucceedsOrMeets	TIMESTAMP(<TimestampPrecision>)	HS_SucceedsOrMeets_T
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>), TIMESTAMP(<TimestampPrecision>)	HS_Intersect_TT
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 part of ISO/IEC 13249 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 Sub 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 correspond 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_CreateHistoryTableType(TableName, TrackedColumns);
  CALL HS_CreateHistoryTable(TableName);
  CALL HS_CreateUpdateTrigger(TableName, TrackedColumns);
  CALL HS_CreateInsertTrigger(TableName, TrackedColumns);
  CALL HS_CreateDeleteTrigger(TableName);
  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 4 – An enabled authorization identifier is defined in ISO/IEC 9075.

- 3) If an error occurred in one of sub-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 AllPKColumns 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 primary key columns
  --
  IF NOT EXISTS(TABLE (HS_GetPrimaryKeys(TableName)))
  THEN
    SIGNAL SQLSTATE '2FF17' SET MESSAGE_TEXT=
      'tracked table has no primary key column';
  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.
-- c) it is not part of the primary key of the specified table.
--
SET AllPKeyColumns = ARRAY(TABLE (HS_GetPrimaryKeys(TableName)));
SET i = 1;
WHILE i <= CARDINALITY(TrackedColumns) DO
    IF TrackedColumns[i] IS NULL THEN
        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
        SIGNAL SQLSTATE '2FF06' SET MESSAGE_TEXT =
            'column does not exist';
    ELSE
        SET j = 1;
        WHILE j <= CARDINALITY(AllPKeyColumns) DO
            IF HS_ExtractColumnIdentifier(TrackedColumns[i]) =
                AllPKeyColumns[j] THEN
                SIGNAL SQLSTATE '2FF18' SET MESSAGE_TEXT =
                    'a primary key column cannot be specified as a tracked column
                    explicitly';
            END IF;
            SET j = j + 1;
        END WHILE;
    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\_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;
  DECLARE tmpAllPKeyColumnsDef CLOB;

  CALL HS_CreateCommaSeparatedTrackedColumnAndTypeList(
    TableName, TrackedColumns, tmpAllTrackedColumnsDef);
  CALL HS_CreateCommaSeparatedPrimaryKeyAndTypeList(
    TableName, tmpAllPKeyColumnsDef);

  SET stmt =
    'CREATE TYPE '
    HS_ConstructTableTypeIdentifier(TableName)
    ' AS(' || tmpAllPKeyColumnsDef
    ' , ' || tmpAllTrackedColumnsDef
    ' , HS_Hist HS_History)'
    ' STATIC METHOD HS_HistoryTable()'
    ' RETURNS TABLE(' || tmpAllPKeyColumnsDef
    ' , ' || tmpAllTrackedColumnsDef
    ' , HS_Hist HS_History),'
    ' STATIC METHOD HS_PNormalize('
    '   targetColumn CHARACTER VARYING(<ColumnNameLength>))'
    ' RETURNS TABLE(' || tmpAllPKeyColumnsDef
    ' , ' || tmpAllTrackedColumnsDef
    ' , HS_Hist HS_History),'
    ' STATIC METHOD HS_PNormalize('
    '   targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)'
    ' RETURNS TABLE(' || tmpAllPKeyColumnsDef
    ' , ' || 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*.

### 5.1.4 HS\_CreateHistoryTable Procedure

#### Purpose

Create a history table corresponding to the specified tracked table.

#### Definition

```

CREATE PROCEDURE HS_CreateHistoryTable
  (IN TableName CHARACTER VARYING(<TableNameLength>))
LANGUAGE SQL
NOT DETERMINISTIC
MODIFIES SQL DATA
BEGIN
  DECLARE stmt CLOB;
  DECLARE tmpAllPKeyColumns CLOB;
  DECLARE tmpAllPKeyColumnsOpt CLOB;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);
  DECLARE tmpTableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);
  SET tmpTableTypeIdentifier = HS_ConstructTableTypeIdentifier(TableName);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, NULL, tmpAllPKeyColumns);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, ' WITH OPTIONS NOT NULL',
    tmpAllPKeyColumnsOpt);

  SET stmt =
    'CREATE TABLE ' || tmpTableIdentifier
    ' OF ' || tmpTableTypeIdentifier
    '(REF IS "HS_REF" SYSTEM GENERATED,'
    tmpAllPKeyColumnsOpt || ','
    'CHECK ('
    '  UNIQUE ('
    '    SELECT ' || tmpAllPKeyColumns || ', '
    '    HS_Hist.HS_BeginTime'
    '  FROM ' || tmpTableIdentifier
    '  ))'
    ');'
  EXECUTE IMMEDIATE stmt;
END

```

#### Description

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

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

## 5.1.5 HS\_CreateUpdateTrigger Procedure

## Purpose

Create a trigger to insert a history row into the history table when the 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 tmpAllPKeyColumns CLOB;
  DECLARE tmpAllPKeyColumns_nwr CLOB;
  DECLARE tmpAllPKeyComparison_HT1_nwr CLOB;
  DECLARE tmpAllPKeyComparison_HT2_nwr CLOB;
  DECLARE tmpAllTrackedColumns CLOB;
  DECLARE tmpAllTrackedColumns_odr CLOB;
  DECLARE tmpAllTrackedColumns_nwr CLOB;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);
  DECLARE tmpTriggerIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);
  SET tmpTriggerIdentifier = HS_ConstructUpdTriggerIdentifier(TableName);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, NULL, tmpAllPKeyColumns);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, 'nwr.', NULL, tmpAllPKeyColumns_nwr);
  CALL HS_CreatePrimaryKeySelfJoinCondition(
    TableName, 'HT1.', 'nwr.', tmpAllPKeyComparison_HT1_nwr);
  CALL HS_CreatePrimaryKeySelfJoinCondition(
    TableName, 'HT2.', 'nwr.', tmpAllPKeyComparison_HT2_nwr);

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

  SET stmt =
    'CREATE TRIGGER ' || tmpTriggerIdentifier
    ' 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'
    ;

  SET stmt = stmt ||
    ' DECLARE CurTS TIMESTAMP(<TimestampPrecision>);'
    ' DECLARE histVal HS_History;'
    ' DECLARE histBegTime TIMESTAMP(<TimestampPrecision>); '

```

```

    ' SET CurTS = HS_GetTransactionTimestamp(); '
SET stmt = stmt ||
    ' SET histVal = NEW HS_History('
    ' CurTS, CAST(NULL AS TIMESTAMP(<TimestampPrecision>)); '
    ' UPDATE ' || tmpTableIdentifier || ' HT1'
    ' SET HT1.HS_Hist.HS_EndTime = CurTS'
    ' WHERE ' || tmpAllPKeyComparison_HT1_nwr
    ' AND HT1.HS_Hist.HS_EndTime IS NULL;'
    ' INSERT INTO ' || tmpTableIdentifier
    '(' || tmpAllPKeyColumns
    ',' || tmpAllTrackedColumns
    ', HS_Hist) '
    ' VALUES (' || tmpAllPKeyColumns_nwr
    ', ' || tmpAllTrackedColumns_nwr
    ', histVal); '
SET stmt = stmt || ' 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*.

## 5.1.6 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;
  DECLARE tmpAllPKeyColumns CLOB;
  DECLARE tmpAllPKeyColumns_nwr CLOB;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);
  DECLARE tmpTriggerIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);
  SET tmpTriggerIdentifier = HS_ConstructInsTriggerIdentifier(TableName);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, NULL, tmpAllPKeyColumns);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, 'nwr.', NULL, tmpAllPKeyColumns_nwr);
  CALL HS_CreateCommaSeparatedTrackedColumnList(
    TrackedColumns, NULL, tmpAllTrackedColumns);
  CALL HS_CreateCommaSeparatedTrackedColumnList(
    TrackedColumns, 'nwr.', tmpAllTrackedColumns_nwr);

  SET stmt =
    'CREATE TRIGGER ' || tmpTriggerIdentifier
    ' AFTER INSERT ON ' || TableName
    ' REFERENCING NEW ROW AS nwr'
    ' FOR EACH ROW'
    ' BEGIN ATOMIC'
    ;
  SET stmt = stmt ||
    ' DECLARE CurTS TIMESTAMP(<TimestampPrecision>);'
    ' DECLARE histVal HS_History;'
    ' SET CurTS = HS_GetTransactionTimestamp();'
    ' SET histVal = NEW HS_History('
    ;
  SET stmt = stmt ||
    'CurTS, CAST(NULL AS TIMESTAMP(<TimestampPrecision>)));'
    ' INSERT INTO ' || tmpTableIdentifier || ' ('
    tmpAllPKeyColumns
    ', ' || tmpAllTrackedColumns
    ', HS_Hist'
    ') VALUES ('
    tmpAllPKeyColumns_nwr
    ', ' || tmpAllTrackedColumns_nwr
    ', histVal);'
    ;
  SET stmt = stmt ||
    ' 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*.

## 5.1.7 HS\_CreateDeleteTrigger Procedure

**Purpose**

Create a trigger to update the end time of the latest history row to CURRENT\_TIMESTAMP value when a row is deleted from the corresponding tracked table.

**Definition**

```

CREATE PROCEDURE HS_CreateDeleteTrigger
  (IN TableName CHARACTER VARYING(<TableNameLength>))
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA
BEGIN
  DECLARE stmt CLOB;
  DECLARE tmpAllPKeyComparison_HT1_odr CLOB;
  DECLARE tmpAllPKeyComparison_HT2_odr CLOB;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);
  DECLARE tmpTriggerIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);
  SET tmpTriggerIdentifier = HS_ConstructDelTriggerIdentifier(TableName);
  CALL HS_CreatePrimaryKeySelfJoinCondition(
    TableName, 'HT1.', 'odr.', tmpAllPKeyComparison_HT1_odr);
  CALL HS_CreatePrimaryKeySelfJoinCondition(
    TableName, 'HT2.', 'odr.', tmpAllPKeyComparison_HT2_odr);

  SET stmt =
    'CREATE TRIGGER ' || tmpTriggerIdentifier
    ' AFTER DELETE ON ' || TableName
    ' REFERENCING OLD ROW AS odr'
    ' FOR EACH ROW'
    ' BEGIN ATOMIC'
    ;
  SET stmt = stmt ||
    ' DECLARE CurTS TIMESTAMP(<TimestampPrecision>);'
    ' SET CurTS = HS_GetTransactionTimestamp();'
    ' UPDATE ' || tmpTableIdentifier || ' HT1'
    ' SET HT1.HS_Hist.HS_EndTime = CurTS'
    ' WHERE ' || tmpAllPKeyComparison_HT1_odr
    ' AND HT1.HS_Hist.HS_EndTime IS NULL;'
    ;
  SET stmt = stmt ||
    ' END'
    ;
  EXECUTE IMMEDIATE stmt;
END

```

**Description**

- 1) The *HS\_CreateDeleteTrigger*(CHARACTER VARYING(<TableNameLength>)) procedure takes the following input parameter:
  - a) a CHARACTER VARYING(<TableNameLength>) value *TableName*.

### 5.1.8 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 tmpAllPKeyColumns CLOB;
  DECLARE tmpAllPKeyColumnsDef CLOB;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);
  DECLARE tmpTableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);
  SET tmpTableTypeIdentifier = HS_ConstructTableTypeIdentifier(TableName);
  CALL HS_CreateCommaSeparatedTrackedColumnList(
    TrackedColumns, NULL, tmpAllTrackedColumns);
  CALL HS_CreateCommaSeparatedTrackedColumnAndTypeList(
    TableName, TrackedColumns, tmpAllTrackedColumnsDef);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, NULL, tmpAllPKeyColumns);
  CALL HS_CreateCommaSeparatedPrimaryKeyAndTypeList(
    TableName, tmpAllPKeyColumnsDef);

  SET stmt =
    'CREATE STATIC METHOD HS_HistoryTable()'
    ' RETURNS TABLE('
    tmpAllPKeyColumnsDef || ', ' || tmpAllTrackedColumnsDef
    ', HS_Hist HS_History)'
    ' FOR ' || tmpTableTypeIdentifier
    ' BEGIN'
    ' RETURN TABLE('
    ' SELECT ' || tmpAllPKeyColumns || ', ' || tmpAllTrackedColumns
    ' , HS_Hist'
    ' FROM ' || tmpTableIdentifier || ') 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*.

## 5.1.9 HS\_CreatePNormalizeMethod Procedure

## Purpose

Create methods to obtain a period-normalized table of the specified columns of 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 tmpAllPKeyColumns CLOB;
  DECLARE tmpAllPKeyColumns_HT1 CLOB;
  DECLARE tmpAllPKeyColumnsDef CLOB;
  DECLARE tmpAllPKeyComparison_HT1_HT2 CLOB;
  DECLARE tmpTableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableTypeIdentifier = HS_ConstructTableTypeIdentifier(TableName);
  CALL HS_CreateCommaSeparatedTrackedColumnList(
    TrackedColumns, NULL, tmpAllTrackedColumns);
  CALL HS_CreateCommaSeparatedTrackedColumnList(
    TrackedColumns, 'HT1.', tmpAllTrackedColumns_HT1);
  CALL HS_CreateCommaSeparatedTrackedColumnAndTypeList(
    TableName, TrackedColumns, tmpAllTrackedColumnsDef);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, NULL, tmpAllPKeyColumns);
  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, 'HT1.', NULL, tmpAllPKeyColumns_HT1);
  CALL HS_CreateCommaSeparatedPrimaryKeyAndTypeList(
    TableName, tmpAllPKeyColumnsDef);
  CALL HS_CreatePrimaryKeySelfJoinCondition(
    TableName, 'HT1.', 'HT2.', tmpAllPKeyComparison_HT1_HT2);

  IF CARDINALITY(TrackedColumns) = 1 THEN
    SET stmt =
      'CREATE STATIC METHOD HS_PNormalize('
      ' targetColumn CHARACTER VARYING(<ColumnNameLength>))'
      ' RETURNS TABLE('
      tmpAllPKeyColumnsDef || ', ' || tmpAllTrackedColumnsDef
      ', HS_Hist HS_History)'
      ' FOR ' || tmpTableTypeIdentifier
      ' BEGIN'
      ' RETURN HS_PNormalize(ARRAY[targetColumn]);'
      ' END'
    EXECUTE IMMEDIATE stmt;

  SET stmt =
    'CREATE STATIC METHOD HS_PNormalize('
    ' targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)'
    ' RETURNS TABLE('
    tmpAllPKeyColumnsDef || ', ' || tmpAllTrackedColumnsDef
    ', HS_Hist HS_History)'
    ' FOR ' || tmpTableTypeIdentifier

```

```

        ' BEGIN'
        ' RETURN HS_HistoryTable();'
        ' END'
EXECUTE IMMEDIATE stmt;
ELSE
SET stmt =
'CREATE STATIC METHOD HS_PNormalize('
' targetColumn CHARACTER VARYING(<ColumnNameLength>))'
' RETURNS TABLE('
tmpAllPKeyColumnsDef || ', ' || tmpAllTrackedColumnsDef
', HS_Hist HS_History)'
' FOR ' || tmpTableTypeIdentifier
' BEGIN'
' RETURN HS_PNormalize(ARRAY[targetColumn]);'
' END'
EXECUTE IMMEDIATE stmt;

SET stmt =
'CREATE STATIC METHOD HS_PNormalize('
' targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)'
' RETURNS TABLE('
tmpAllPKeyColumnsDef || ', ' || tmpAllTrackedColumnsDef
', HS_Hist HS_History)'
' FOR ' || tmpTableTypeIdentifier
' BEGIN'
SET stmt = stmt ||
' DECLARE m_tableTypeIdentifier'
' CHARACTER VARYING(<IdentifierLength>);'
' DECLARE m_periodColumnsComparison_HT1_HT2 CLOB;'
' DECLARE m_periodColumns CLOB;'
' DECLARE m_periodColumns_HT1 CLOB;'
' DECLARE m_periodColumns_HT2 CLOB;'
' DECLARE m_tmpAllColumns CLOB;'
' DECLARE m_tmpAllColumns_HT1 CLOB;'
' DECLARE m_tmpAllPKeyColumns CLOB;'
' DECLARE m_tmpPKeyComparison_HT1_HT2 CLOB;'
' DECLARE m_stmt CLOB;'
' DECLARE m_setStmt CLOB;'
' DECLARE m_Result_Tbl ROW('
tmpAllPKeyColumnsDef || ', ' || tmpAllTrackedColumnsDef
', HS_Hist HS_History) MULTISSET;'
' DECLARE m_i INTEGER;'
' DECLARE m_restColumns CLOB;'
' DECLARE m_dlmString CHAR(2) DEFAULT ',';'
' DECLARE m_empString CHAR VARYING(1) DEFAULT '';'
' DECLARE m_endPosition INTEGER;'
' DECLARE m_columnName CLOB;'
' DECLARE m_tmpTargetColumns '
' CHARACTER VARYING(<ColumnNameLength>) ARRAY;'
' SET m_tableTypeIdentifier = '
' '' || tmpTableTypeIdentifier || '';'
' SET m_tmpAllColumns = '' || tmpAllPKeyColumns
', ' || tmpAllTrackedColumns || '';'
' SET m_tmpAllColumns_HT1 = '' || tmpAllPKeyColumns_HT1
', ' || tmpAllTrackedColumns_HT1 || '';'
' SET m_tmpAllPKeyColumns = '' || tmpAllPKeyColumns || '';'
' SET m_tmpPKeyComparison_HT1_HT2 = ''
tmpAllPKeyComparison_HT1_HT2 || '';'
' SET m_periodColumns = '';'
' SET m_periodColumns_HT1 = '';'
' SET m_periodColumns_HT2 = '';'

```

```

' SET m_i = 1;'
' WHILE m_i <= CARDINALITY(targetColumns) DO'
'   SET m_tmpTargetColumns[m_i] = '
'     HS_ConstructColumnIdentifier(targetColumns[m_i]);'
'   SET m_i = m_i + 1;'
' END WHILE;'
' SET m_i = 1;'
' WHILE m_i <= CARDINALITY(targetColumns) DO'
'   IF m_i > 1 THEN'
'     SET m_periodColumns_HT1 = m_periodColumns_HT1 || ', ' ;'
'     SET m_periodColumns_HT2 = m_periodColumns_HT2 || ', ' ;'
'   END IF;'
'   SET m_periodColumns_HT1 = m_periodColumns_HT1 || '
'     'HT1.' || targetColumns[m_i];'
'   SET m_periodColumns_HT2 = m_periodColumns_HT2 || '
'     'HT2.' || targetColumns[m_i];'
'   SET m_i = m_i + 1;'
' END WHILE;'
' SET m_periodColumnsComparison_HT1_HT2 = '
'   '(' || m_periodColumns_HT1 || ') ' || '
'   'IS NOT DISTINCT FROM (' || m_periodColumns_HT2 || ')';'
' SET m_restColumns = m_tmpAllColumns || m_dlmString;'
' WHILE m_restColumns <> m_empString DO'
'   SET m_endPosition = '
'     POSITION(m_dlmString IN m_restColumns) - 1;'
'   SET m_columnName = '
'     SUBSTRING(m_restColumns FROM 1 FOR m_endPosition);'
'   IF POSITION(m_columnName || m_dlmString'
'     IN m_tmpAllPKeyColumns || m_dlmString) >= 1 THEN'
'     SET m_periodColumns = m_periodColumns'
'       || m_columnName || ', ' ;'
'   ELSEIF NOT EXISTS(SELECT * '
'     FROM UNNEST(m_tmpTargetColumns) AS CLMS(CNAME) '
'     WHERE CNAME = m_columnName) THEN'
'     SET m_periodColumns = m_periodColumns || '
'       'NULLIF(' || m_columnName || ', ' '
'       || m_columnName || ') ' || ', ' ;'
'   ELSE'
'     SET m_periodColumns = m_periodColumns || '
'       m_columnName || ', ' ;'
'   END IF;'
'   SET m_restColumns = '
'     OVERLAY(m_restColumns PLACING m_empString '
'     FROM 1 FOR m_endPosition + CHAR_LENGTH(m_dlmString));'
' END WHILE;'
' SET m_stmt = '
'   'WITH RECURSIVE PeriodTemp(' || '
'     m_tmpAllColumns || ', HS_Hist) AS ' || ' (' || '
'     SELECT ' || m_periodColumns || 'HS_Hist' || '
'     FROM TABLE(' || m_tableTypeIdentifier || '
'       '::HS_HistoryTable()) AS HT' || '
'     UNION ALL ' || '
'     SELECT ' || m_tmpAllColumns_HT1 || ', ' || '
'     NEW HS_History(' || '
'       HT1.HS_Hist.HS_BeginTime, ' || '
'       HT2.HS_Hist.HS_EndTime)' || '
'     FROM PeriodTemp HT1, TABLE(' || m_tableTypeIdentifier || '
'       '::HS_HistoryTable()) HT2' || '
'     WHERE ' || m_tmpPKeyComparison_HT1_HT2 || '
'     AND HT1.HS_Hist.HS_EndTime = ' || '
'     HT2.HS_Hist.HS_BeginTime' || '

```

```

'    '    AND ('||m_periodColumnsComparison_HT1_HT2||')''||'
'    '    )''||'
'    '    SELECT '||m_tmpAllColumns_HT1||', HT1.HS_Hist''||'
'    '    FROM PeriodTemp HT1''||'
'    '    WHERE NOT EXISTS('||'
'    '    SELECT *''||'
'    '    FROM PeriodTemp HT2''||'
'    '    WHERE '||m_tmpPKeyComparison_HT1_HT2||'
'    '    AND ('||m_periodColumnsComparison_HT1_HT2||')''||'
'    '    AND ('||'
'    '    ('||'
'    '    HT2.HS_Hist.HS_EndTime IS NOT NULL''||'
'    '    AND''||'('||'
'    '    HT2.HS_Hist.HS_Contains('||'
'    '    HT1.HS_Hist.HS_BeginTime) = 1'' ||'
'    '    AND'' ||'
'    '    HT2.HS_Hist.HS_BeginTime '' ||'
'    '    <> HT1.HS_Hist.HS_BeginTime'' ||'
'    '    OR'' ||'
'    '    HT2.HS_Hist.HS_Contains('||'
'    '    HT1.HS_Hist.HS_EndTime) = 1'' ||'
'    '    AND'' ||'
'    '    HT2.HS_Hist.HS_BeginTime '' ||'
'    '    <> HT1.HS_Hist.HS_EndTime'' ||'
'    '    )''||'
'    '    )''||' OR ''||'('||'
'    '    HT2.HS_Hist.HS_EndTime IS NULL''||'
'    '    AND ''||'('||'
'    '    HT2.HS_Hist.HS_BeginTime''||'
'    '    < HT1.HS_Hist.HS_BeginTime''||'
'    '    OR''||'
'    '    HT2.HS_Hist.HS_BeginTime''||'
'    '    < HT1.HS_Hist.HS_EndTime''||'
'    '    )''||'
'    '    )''||'
'    '    )''||'
'    '    )'';'
;

SET stmt = stmt ||
' SET m_setStmt = 'SET ? = TABLE('||m_stmt||')'';'
' PREPARE pstmt FROM m_setStmt;'
' EXECUTE pstmt INTO m_Result_Tbl;'
' RETURN m_Result_Tbl;'
;

SET stmt = stmt ||
' END'
;

EXECUTE IMMEDIATE stmt;
END IF;
END

```

## Description

- 1) The definition of *HS\_PNormalize* method of *<TableTypeIdIdentifier>* 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*.

5.1.10 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 tmpAllPKeyColumns CLOB;
  DECLARE tmpAllTrackedColumns CLOB;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);

  CALL HS_CreateCommaSeparatedPrimaryKeyList(
    TableName, NULL, NULL, tmpAllPKeyColumns);
  CALL HS_CreateCommaSeparatedTrackedColumnList(
    TrackedColumns, NULL, tmpAllTrackedColumns);

  SET stmt =
    'INSERT INTO ' || tmpTableIdentifier
    '(' || tmpAllPKeyColumns
    ', ' || tmpAllTrackedColumns
    ', HS_Hist)'
    ' SELECT '
    tmpAllPKeyColumns
    ', ' || tmpAllTrackedColumns
    ', NEW HS_History('
  SET stmt = stmt ||
    '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 Sub 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);
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
  IF TableName IS NULL THEN
    SIGNAL SQLSTATE '2FF01' SET MESSAGE_TEXT =
      'table name is a null value';
  END IF;
  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;
  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 which <TableNameIdentifier> type is created in.
- 2) <SchemaName> is <unqualified schema name> of the schema which <TableNameIdentifier> type is created in.

#### 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;
  DECLARE tmpTriggerIdentifier CHARACTER VARYING(<IdentifierLength>);

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

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

  --
  -- Drop UPDATE trigger if exist
  --
  SET tmpTriggerIdentifier = HS_ConstructUpdTriggerIdentifier(TableName);
  IF EXISTS (
    SELECT *
    FROM INFORMATION_SCHEMA.TRIGGERS
    WHERE TRIGGER_CATALOG = <CatalogName>
      AND TRIGGER_SCHEMA = <SchemaName>
      AND TRIGGER_NAME = tmpTriggerIdentifier
    ) THEN
    SET stmt =

```

```
        'DROP TRIGGER ' || tmpTriggerIdentifier          ;
EXECUTE IMMEDIATE stmt;
END IF;
END
```

**Definitional Rules**

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

**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;
  DECLARE tmpTableIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableIdentifier = HS_ConstructTableIdentifier(TableName);
  IF EXISTS (
    SELECT *
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_CATALOG = <CatalogName>
      AND TABLE_SCHEMA = <SchemaName>
      AND TABLE_NAME = tmpTableIdentifier
    ) THEN
    SET stmt =
      'DROP TABLE ' || tmpTableIdentifier
    EXECUTE IMMEDIATE stmt;
  END IF;
END

```

### Definitional Rules

- 1) <CatalogName> is <catalog name> of the schema which history table is created in.
- 2) <SchemaName> is <unqualified schema name> of the schema which history table is created in.

### 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;
  DECLARE tmpTableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);

  SET tmpTableTypeIdentifier = HS_ConstructTableTypeIdentifier(TableName);
  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 = tmpTableTypeIdentifier
    ) THEN
    SET stmt =
      'DROP TYPE ' || tmpTableTypeIdentifier
    EXECUTE IMMEDIATE stmt;
  END IF;
END

```

### Definitional Rules

- 1) <CatalogName> is <catalog name> of the schema which <TableTypeIdentifier> type is created in.
- 2) <SchemaName> is <unqualified schema name> of the schema which <TableTypeIdentifier> type is created in.

### Description

- 1) The *HS\_DropHistoryTableType*(CHARACTER VARYING(<TableNameLength>)) procedure takes the following input parameter:
  - a) a CHARACTER VARYING(<TableNameLength>) value *TableName*.

### 5.3 Utility Procedures for History

The procedures in this subsection execute process performed in sub-procedure of the HS\_CreateHistory procedure or the HS\_DropHistory procedure.

#### 5.3.1 HS\_CreateCommaSeparatedPrimaryKeyList Procedure

##### Purpose

For all primary key columns of the specified table, generate comma-separated list of the text which concatenates the specified prefix, the name of column of the specified table and the specified postfix.

##### Definition

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

  DECLARE i INTEGER;

  SET AllPKeyColumns = ARRAY(TABLE (HS_GetPrimaryKeys(TableName)));

  SET i = 1;
  SET ResultValue = '';
  WHILE i <= CARDINALITY(AllPKeyColumns) DO
    IF i > 1 THEN
      SET ResultValue = ResultValue || ', ' ;
    END IF;
    IF ColumnNamePrefix IS NOT NULL THEN
      SET ResultValue = ResultValue || ColumnNamePrefix ;
    END IF;
    SET ResultValue = ResultValue ||
      HS_ConstructColumnIdentifier(AllPKeyColumns[i]) ;
    IF ColumnNamePostfix IS NOT NULL THEN
      SET ResultValue = ResultValue || ColumnNamePostfix ;
    END IF;
    SET i = i + 1;
  END WHILE;
END
```

##### Description

- 1) The *HS\_CreateCommaSeparatedPrimaryKeyList*(CHARACTER VARYING(<TableNameLength>), CLOB, CLOB, CLOB) procedure takes the following input parameters:
  - a) a CHARACTER VARYING(<TableNameLength>) value *TableName*,
  - b) a CLOB value *ColumnNamePrefix*,
  - c) a CLOB value *ColumnNamePostfix*.
- 2) The *HS\_CreateCommaSeparatedPrimaryKeyList*(CHARACTER VARYING(<TableNameLength>), CLOB, CLOB, CLOB) procedure takes the following output parameter:

- a) a CLOB value *ResultValue*.
- 3) For the procedure *HS\_CreateCommaSeparatedPrimaryKeyList*(*CHARACTER VARYING*(<*TableNameLength*>), *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) 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).
- 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 *CN<sub>n</sub>* be the <column name> represented by the n-th element of array which is a return value of *HS\_GetPrimaryKeys*(*CHARACTER VARYING*(<*TableNameLength*>) function.
- e) Let *CID<sub>n</sub>* be the value of the result of *HS\_ConstructColumnIdentifier*(*CN<sub>n</sub>*). Let *PID<sub>n</sub>* be the character string value of a concatenation:
- <quote>*PRX*<quote> || *CID<sub>n</sub>* || <quote>*PTX*<quote>
- f) Let *RV* be the character string value of a concatenation:
- PID<sub>1</sub>* || <quote><comma><quote> || *PID<sub>2</sub>* || <quote><comma><quote> || ... || *PID<sub>n</sub>*
- g) Set the value of output parameter *ResultValue* to *RV*.

### 5.3.2 HS\_CreateCommaSeparatedPrimaryKeyAndTypeList Procedure

#### Purpose

For all primary key columns of the specified table, generate comma-separated list of the text which concatenates the specified prefix, the name of column of the specified table, white space, the name of data type or the name of domain and the specified postfix.

#### Definition

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

  DECLARE i INTEGER;

  --
  -- !! See Description
  --      about "AllPKeyColumnsAndTypes"
  --

  SET i = 1;
  SET ResultValue = '';
  WHILE i <= CARDINALITY(AllPKeyColumnsAndTypes) DO
    IF i > 1 THEN
      SET ResultValue = ResultValue || ', ' ;
    END IF;
    SET ResultValue = ResultValue || AllPKeyColumnsAndTypes[i] ;
    SET i = i + 1;
  END WHILE;
END
```

#### Description

- 1) The *HS\_CreateCommaSeparatedPrimaryKeyAndTypeList*(*CHARACTER VARYING*(<*TableNameLength*>), *CLOB*) procedure takes the following input parameter:
  - a) a *CHARACTER VARYING*(<*TableNameLength*>) value *TableName*.
- 2) The *HS\_CreateCommaSeparatedPrimaryKeyAndTypeList*(*CHARACTER VARYING*(<*TableNameLength*>), *CLOB*) procedure takes the following output parameter:
  - a) a *CLOB* value *ResultValue*.
- 3) For the procedure *HS\_CreateCommaSeparatedPrimaryKeyAndTypeList*(*CHARACTER VARYING*(<*TableNameLength*>), *CLOB*):
  - a) The value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.
  - b) Let *CN<sub>n</sub>* be the <column name> represented by the n-th element of array which is a return value of *HS\_GetPrimaryKeys*(*CHARACTER VARYING*(<*TableNameLength*>) function.
  - c) Let *CID<sub>n</sub>* be the value of the result of *HS\_ConstructColumnIdentifier*(*CN<sub>n</sub>*).
  - d) Let *DT<sub>n</sub>* be the <data type or domain name> which correspond to the column *CN<sub>n</sub>*.

e) Let  $CDN_n$  be the character string value of a concatenation:

$$CID_n \parallel \langle \text{quote} \rangle \langle \text{space} \rangle \langle \text{quote} \rangle \parallel \langle \text{quote} \rangle DT_n \langle \text{quote} \rangle$$

f) Set the value of n-th element of array  $AllPKeyColumnsAndTypes$  to  $CDN_n$ .

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

$$CDN_1 \parallel \langle \text{quote} \rangle \langle \text{comma} \rangle \langle \text{quote} \rangle \parallel CDN_2 \parallel \langle \text{quote} \rangle \langle \text{comma} \rangle \langle \text{quote} \rangle \parallel \dots \parallel CDN_n$$

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

### 5.3.3 HS\_CreatePrimaryKeySelfJoinCondition Procedure

#### Purpose

For all primary key columns of the specified table, generate the text of self-join condition for the specified table. The text is a list of equal predicate separated by ' AND'. Each equal predicate is a text which concatenates the specified prefix text, the name of column, an equal sign, another specified prefix text and the name of column.

#### Definition

```

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

  DECLARE i INTEGER;

  SET AllPKeyColumns = ARRAY(TABLE (HS_GetPrimaryKeys(TableName)));

  SET i = 1;
  SET ResultValue = '';
  WHILE i <= CARDINALITY(AllPKeyColumns) DO
    IF i > 1 THEN
      SET ResultValue = ResultValue || ' AND ' ;
    END IF;
    IF ColumnNamePrefix1 IS NOT NULL THEN
      SET ResultValue = ResultValue || ColumnNamePrefix1 ;
    END IF;
    SET ResultValue = ResultValue ||
      HS_ConstructColumnIdentifier(AllPKeyColumns[i]) ;

    SET ResultValue = ResultValue || ' = ' ;

    IF ColumnNamePrefix2 IS NOT NULL THEN
      SET ResultValue = ResultValue || ColumnNamePrefix2 ;
    END IF;
    SET ResultValue = ResultValue ||
      HS_ConstructColumnIdentifier(AllPKeyColumns[i]) ;

    SET i = i + 1;
  END WHILE;
END

```

#### Description

- 1) The *HS\_CreatePrimaryKeySelfJoinCondition*(CHARACTER VARYING(<TableNameLength>), CLOB, CLOB, CLOB) procedure takes the following input parameters:
  - a) a CHARACTER VARYING(<TableNameLength>) value *TableName*,
  - b) a CLOB value *ColumnNamePrefix1*,
  - c) a CLOB value *ColumnNamePrefix2*.
- 2) The *HS\_CreatePrimaryKeySelfJoinCondition*(CHARACTER VARYING(<TableNameLength>), CLOB, CLOB, CLOB) procedure takes the following output parameter:

- a) a CLOB value *ResultValue*.
- 3) For the procedure *HS\_CreatePrimaryKeySelfJoinCondition*(*CHARACTER VARYING*(<*TableNameLength*>), *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) 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).
- c) 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).
- d) Let *CN<sub>n</sub>* be the <column name> represented by the n-th element of array which is a return value of *HS\_GetPrimaryKeys*(*CHARACTER VARYING*(<*TableNameLength*>) function.
- e) Let *CID<sub>n</sub>* be the value of the result of *HS\_ConstructColumnIdentifier*(*CN<sub>n</sub>*). Let *CND<sub>n</sub>* be the character string value of a concatenation:
- <quote>*PRX1*<quote> || *CID<sub>n</sub>* || <quote><equals operator><quote> || <quote>*PRX2*<quote> || *CID<sub>n</sub>*
- f) Let *RV* be the character string value of a concatenation:
- CND1* || <quote>*AND*<quote> || *CND2* || <quote>*AND*<quote> || ... || *CND<sub>n</sub>*
- g) Set the value of output parameter *ResultValue* to *RV*.

### 5.3.4 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:

<quote>*RV*<quote>

### 5.3.5 HS\_CreateCommaSeparatedTrackedColumnList Procedure

#### Purpose

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

#### 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;

  SET i = 1;
  SET ResultValue = '';
  WHILE i <= CARDINALITY(TrackedColumns) DO
    IF i > 1 THEN
      SET ResultValue = ResultValue || ', ' ;
    END IF;
    IF ColumnNamePrefix IS NOT NULL THEN
      SET ResultValue = ResultValue || ColumnNamePrefix ;
    END IF;
    SET ResultValue = ResultValue ||
      HS_ConstructColumnIdentifier(TrackedColumns[i]) ;
    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 *CN<sub>n</sub>* 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 *CID<sub>n</sub>* be the value of the result of *HS\_ConstructColumnIdentifier(CN<sub>n</sub>)*. Let *PID<sub>n</sub>* 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*.

### 5.3.6 HS\_CreateCommaSeparatedTrackedColumnAndTypeList Procedure

#### Purpose

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

#### 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;

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

  SET i = 1;
  SET ResultValue = '';
  WHILE i <= CARDINALITY(TrackedColumnsAndTypes) DO
    IF i > 1 THEN
      SET ResultValue = ResultValue || ', '           ;
    END IF;
    SET ResultValue = ResultValue || TrackedColumnsAndTypes[i] ;
    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 *CN<sub>n</sub>* be the <column name> represented by the n-th element of the array of parameter *TrackedColumns*.
- c) Let *CID<sub>n</sub>* be the value of the result of *HS\_ConstructColumnIdentifier(CN<sub>n</sub>)*.
- d) Let *DT<sub>n</sub>* be the <data type or domain name> which corresponds to the column *CN<sub>n</sub>*.
- e) Let *CDN<sub>n</sub>* be the character string value of a concatenation:  
$$CID_n \parallel \langle \text{quote} \rangle \langle \text{space} \rangle \langle \text{quote} \rangle \parallel \langle \text{quote} \rangle DT_n \langle \text{quote} \rangle$$
- f) Set the value of n-th element of array *TrackedColumnsAndTypes* to *CDN<sub>n</sub>*.
- g) Let *RV* be the character string value of a concatenation:  
$$CDN_1 \parallel \langle \text{quote} \rangle \langle \text{comma} \rangle \langle \text{quote} \rangle \parallel CDN_2 \parallel \langle \text{quote} \rangle \langle \text{comma} \rangle \langle \text{quote} \rangle \parallel \dots \parallel CDN_n$$
- h) Set the value of output parameter *ResultValue* to *RV*.

### 5.3.7 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(<TableNameLength>),
   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 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 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 HS_ConstructIdentifier('', ColumnName, '');
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 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 HS_ConstructIdentifier('HS_TR_', TableName, '_DEL');
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 HS_ConstructIdentifier('HS_TR_', TableName, '_INS');
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\_ConstructUpdTriggerIdentifier*(*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\_ConstructDelTriggerIdentifier*(*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\_ConstructInsTriggerIdentifier*(*CHARACTER VARYING*(<*TableNameLength*>)), the value of parameter *TableName* is a character representation of a table name which forms an instance of <table name>.

### 5.3.8 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>.

### 5.3.9 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.

## 5.4 <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> is created in.

### Definitional Rules

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

NOTE 5 – Other objects than <TableTypeIdentifier> and its methods, such as a history table and triggers, may also be created in this schema. However, this part of ISO/IEC 13249 requires only <TableTypeIdentifier> and its methods for the conformance.

NOTE 6 – This schema may or may not be identical to the schema that includes the tracked table whose history is being defined.

- 2) <CatalogName> is equivalent to <catalog name> contained in <TableName>.
- 3) <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>.
- 4) <catalog name> and <unqualified schema name> of the schema provided for <TableTypeIdentifier> 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, a schema, 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
  (timePoint TIMESTAMP(<TimestampPrecision>))
  RETURNS INTEGER,

METHOD HS_Meets
  (hs_hist HS_History)
  RETURNS INTEGER,

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

METHOD HS_Precedes
  (hs_hist HS_History)
  RETURNS INTEGER,

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

METHOD HS_PrecedesOrMeets
  (hs_hist HS_History)
  RETURNS INTEGER,

METHOD HS_Succeeds
  (timePoint TIMESTAMP(<TimestampPrecision>))

```

```

    RETURNS INTEGER,

METHOD HS_Succeeds
    (hs_hist HS_History)
    RETURNS INTEGER,

METHOD HS_SucceedsOrMeets
    (timePoint 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*>)),
- d) a method *HS\_Meets*(*HS\_History*),
- e) a method *HS\_Precedes*(*TIMESTAMP*(<*TimestampPrecision*>)),
- f) a method *HS\_Precedes*(*HS\_History*),
- g) a method *HS\_PrecedesOrMeets*(*TIMESTAMP*(<*TimestampPrecision*>)),
- h) a method *HS\_PrecedesOrMeets*(*HS\_History*),
- i) a method *HS\_Succeeds*(*TIMESTAMP*(<*TimestampPrecision*>)),
- j) a method *HS\_Succeeds*(*HS\_History*),
- k) a method *HS\_SucceedsOrMeets*(*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 beginOfPeriod IS NULL THEN
      SIGNAL SQLSTATE '2FF09' SET MESSAGE_TEXT =
        'beginning of period is a null value';
    ELSEIF endOfPeriod < beginOfPeriod THEN
      SIGNAL SQLSTATE '2FF10' SET MESSAGE_TEXT =
        'end of period precedes beginning of period';
    ELSEIF endOfPeriod = beginOfPeriod THEN
      SIGNAL SQLSTATE '2FF16' SET MESSAGE_TEXT =
        'empty period';
    ELSEIF HS_GetTransactionTimestamp() <= beginOfPeriod THEN
      SIGNAL SQLSTATE '2FF23' SET MESSAGE_TEXT =
        'beginning of period succeeds current timestamp';
    ELSEIF HS_GetTransactionTimestamp() <= endOfPeriod THEN
      SIGNAL SQLSTATE '2FF24' SET MESSAGE_TEXT =
        'end of period succeeds current timestamp';
    END IF;
    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
    DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    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 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
            1
          ELSE
            0
        END;
    ELSEIF s_et IS NULL AND hs_hist.HS_EndTime IS NULL THEN
      --
      -- Both end time are null
      --
      RETURN 1;
    ELSEIF s_et IS NULL THEN
      --
      -- Only an end time of SELF 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
            1
          ELSE
            0
        END;
    ELSE
      --

```

```

-- Only an end time of given period 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
        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
  (timePoint TIMESTAMP(<TimestampPrecision>))
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    SET CurTS = HS_GetTransactionTimestamp();

    --
    -- If a given timestamp value expresses future,
    -- 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_et = SELF.HS_EndTime;

    RETURN
      CASE
        WHEN (s_et = timePoint) THEN
          1
        ELSE
          0
      END;
  END
END

CREATE METHOD HS_Meets
  (hs_hist HS_History)
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    RETURN SELF.HS_Meets(hs_hist.HS_BeginTime);
  END
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>)) method takes the following input parameter:
  - a) a TIMESTAMP(<TimestampPrecision>) value *timePoint*.
- 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
  (timePoint TIMESTAMP(<TimestampPrecision>))
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    SET CurTS = HS_GetTransactionTimestamp();

    --
    -- If a given timestamp value expresses future,
    -- 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_et = SELF.HS_EndTime;

    RETURN
      CASE
        WHEN (s_et < timePoint) THEN
          1
        ELSE
          0
      END;
  END
END

CREATE METHOD HS_Precedes
  (hs_hist HS_History)
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    RETURN SELF.HS_Precedes(hs_hist.HS_BeginTime);
  END
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>)) method takes the following input parameter:
  - a) a TIMESTAMP(<TimestampPrecision>) value *timePoint*.
- 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
  (timePoint TIMESTAMP(<TimestampPrecision>))
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    SET CurTS = HS_GetTransactionTimestamp();

    --
    -- If a given timestamp value expresses future, then signal an error.
    --
    IF CurTS < timePoint THEN
      SIGNAL SQLSTATE '2FF25' SET MESSAGE_TEXT =
        'a given timestamp value expresses future time';
    END IF;

    IF SELF.HS_Precedes(timePoint) = 1 THEN
      RETURN 1;
    ELSEIF SELF.HS_Meets(timePoint) = 1 THEN
      RETURN 1;
    ELSE
      RETURN 0;
    END IF;
  END
END

CREATE METHOD HS_PrecedesOrMeets
  (hs_hist HS_History)
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    IF SELF.HS_Precedes(hs_hist) = 1 THEN
      RETURN 1;
    ELSEIF SELF.HS_Meets(hs_hist) = 1 THEN
      RETURN 1;
    ELSE
      RETURN 0;
    END IF;
  END
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>))* method takes the following input parameter:
  - a) a *TIMESTAMP(<TimestampPrecision>)* value *timePoint*.
- 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
  (timePoint TIMESTAMP(<TimestampPrecision>))
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    SET CurTS = HS_GetTransactionTimestamp();

    --
    -- If a given timestamp value expresses future,
    -- then signal an error.
    --
    IF CurTS < timePoint THEN
      SIGNAL SQLSTATE '2FF25' SET MESSAGE_TEXT =
        'a given timestamp value expresses future time';
    END IF;

    RETURN
      CASE
        WHEN (timePoint < SELF.HS_BeginTime) THEN
          1
        ELSE
          0
      END;
  END

CREATE METHOD HS_Succeeds
  (hs_hist HS_History)
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    RETURN SELF.HS_Succeeds(
      hs_hist.HS_EndTime);
  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*>)) method takes the following input parameter:
  - a) a *TIMESTAMP*(<*TimestampPrecision*>) value *timePoint*.
- 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 is met by the specified period.

#### Definition

```

CREATE METHOD HS_SucceedsOrMeets
  (timePoint TIMESTAMP(<TimestampPrecision>))
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    SET CurTS = HS_GetTransactionTimestamp();

    --
    -- If a given timestamp value expresses future, then signal an error.
    --
    IF CurTS < timePoint THEN
      SIGNAL SQLSTATE '2FF25' SET MESSAGE_TEXT =
        'a given timestamp value expresses future time';
    END IF;

    IF SELF.HS_Succeeds(timePoint) = 1 THEN
      RETURN 1;
    ELSEIF timePoint = SELF.HS_BeginTime THEN
      RETURN 1;
    ELSE
      RETURN 0;
    END IF;
  END
END

CREATE METHOD HS_SucceedsOrMeets
  (hs_hist HS_History)
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    IF SELF.HS_Succeeds(hs_hist) = 1 THEN
      RETURN 1;
    ELSEIF hs_hist.HS_Meets(SELF) = 1 THEN
      RETURN 1;
    ELSE
      RETURN 0;
    END IF;
  END
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>))* method takes the following input parameter:
  - a) a *TIMESTAMP(<TimestampPrecision>)* value *timePoint*.
- 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 expresses future,
    -- 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
          1
        WHEN NOT (s_bt <= timePoint AND timePoint < s_et) THEN
          0
        ELSE
          CAST(NULL AS INTEGER)
      END;
  END
END

CREATE METHOD HS_Contains
  (hs_hist HS_History)
  RETURNS INTEGER
  FOR HS_History
  BEGIN
    DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);
    DECLARE p_bt TIMESTAMP(<TimestampPrecision>);
    DECLARE p_et TIMESTAMP(<TimestampPrecision>);
    DECLARE CurTS TIMESTAMP(<TimestampPrecision>);

    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
      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
      0
    ELSE
      CAST(NULL AS INTEGER)
  END;
ELSE
  RETURN
  CASE
    WHEN (s_bt <= p_bt AND p_et <= CurTS) THEN
      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
    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
          1
        WHEN NOT (s_bt = p_bt AND s_et = p_et) THEN
          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 year-month interval.

#### Definition

```
CREATE METHOD HS_MonthInterval()  
  RETURNS INTERVAL YEAR TO MONTH  
  FOR HS_History  
  BEGIN  
    DECLARE s_bt TIMESTAMP(<TimestampPrecision>);  
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);  
  
    SET s_bt = SELF.HS_BeginTime;  
    SET s_et = SELF.HS_EndTime;  
  
    IF s_et IS NULL THEN  
      RETURN CAST(NULL AS INTEGER);  
    END IF;  
  
    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 day-time interval.

#### Definition

```
CREATE METHOD HS_DayInterval()  
  RETURNS INTERVAL DAY TO SECOND  
  FOR HS_History  
  BEGIN  
    DECLARE s_bt TIMESTAMP(<TimestampPrecision>);  
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);  
  
    SET s_bt = SELF.HS_BeginTime;  
    SET s_et = SELF.HS_EndTime;  
  
    IF s_et IS NULL THEN  
      RETURN CAST(NULL AS INTEGER);  
    END IF;  
  
    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 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
    DECLARE s_bt TIMESTAMP(<TimestampPrecision>);
    DECLARE s_et TIMESTAMP(<TimestampPrecision>);
    DECLARE p_bt TIMESTAMP(<TimestampPrecision>);
    DECLARE p_et TIMESTAMP(<TimestampPrecision>);

    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;

    SET st = CASE
      WHEN p_bt < s_bt THEN s_bt
      ELSE p_bt
    END;
    SET et = CASE
      WHEN s_et < p_et THEN s_et
      WHEN p_et <= s_et THEN p_et
      ELSE NULL
    END;
    IF s_et IS NULL AND p_et IS NULL THEN
      RETURN NEW HS_History(st, CAST(NULL AS TIMESTAMP(<TimestampPrecision>)));
    ELSEIF st < et THEN
      SET resultHSHist = NEW HS_History(
        st, et);
      RETURN resultHSHist;
    ELSEIF et IS NULL THEN
      --
      -- One of s_et or p_et has NULL value
      --
      SET resultHSHist = NEW HS_History(
        st, COALESCE(s_et, p_et));
      RETURN resultHSHist;
    ELSE
      SIGNAL SQLSTATE '2FF19' SET MESSAGE_TEXT =
        'result of the intersect operation is empty period';
    END IF;
  END
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 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
    IF SELF.HS_EndTime < hs_hist.HS_BeginTime
      OR hs_hist.HS_EndTime < SELF.HS_BeginTime THEN
      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;
        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 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
    DECLARE resultHSHist HS_History;
    IF SELF.HS_EndTime <= hs_hist.HS_BeginTime
      OR hs_hist.HS_EndTime <= SELF.HS_BeginTime THEN
      RETURN SELF;
    ELSEIF SELF.HS_BeginTime < hs_hist.HS_BeginTime
      AND (SELF.HS_EndTime <= hs_hist.HS_EndTime
        OR hs_hist.HS_EndTime IS NULL) THEN
      SET resultHSHist = NEW HS_History(
        SELF.HS_BeginTime, hs_hist.HS_BeginTime);
      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
      SET resultHSHist = NEW HS_History(
        hs_hist.HS_EndTime, SELF.HS_EndTime);
      RETURN resultHSHist;
    ELSEIF SELF.HS_BeginTime < hs_hist.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
      SIGNAL SQLSTATE '2FF21' SET MESSAGE_TEXT =
        'result of the except operation has disjoint periods';
    ELSE
      SIGNAL SQLSTATE '2FF22' SET MESSAGE_TEXT =
        'result of the except operation is empty period';
    END IF;
  END
END

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

```

**Description**

- 1) The *HS\_Except*(*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\_Except(HS\_History)* method takes the following input parameter:
  - a) an *HS\_History* value *hs\_hist*.

## 6.2 <TableTypeIdentifier> Type and Routines

### 6.2.1 <TableTypeIdentifier> Type

#### Purpose

Provide the definition of history row corresponding to the specified tracked table.

#### Definition

```
CREATE TYPE <TableTypeIdentifier>
AS (
  <AllPKeyColumnsDef>,
  <AllTrackedColumnsDef>,
  HS_Hist HS_History
)

STATIC METHOD HS_HistoryTable()
RETURNS TABLE(
  <AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History),

STATIC METHOD HS_PNormalize
(targetColumn CHARACTER VARYING(<ColumnNameLength>))
RETURNS TABLE(
  <AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History),

STATIC METHOD HS_PNormalize
(targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
RETURNS TABLE(
  <AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History),
```

#### Description

- 1) Definition of this method uses the following substitutes.
  - a) <AllPKeyColumnsDef>: Comma-separated character strings which concatenate the primary key column name of a tracked table and the name of data type of the column.
  - b) <AllTrackedColumnsDef>: Comma-separated character strings which concatenate the tracked column name of a tracked table and the name of data type of the column.
  - c) <TableTypeIdentifier>: The value of the result of invoking *HS\_ConstructTableTypeIdentifier* with the name of a tracked table as its argument.
- 2) The <TableTypeIdentifier> type provides the following methods for public use:
  - a) a method *HS\_HistoryTable()*,
  - b) a method *HS\_PNormalize(CHARACTER VARYING(<ColumnNameLength>))*,
  - c) a method *HS\_PNormalize(CHARACTER VARYING(<ColumnNameLength>) ARRAY)*.
- 3) The <TableTypeIdentifier> type has the following attributes:
  - a) attributes correspond to primary key columns of tracked table,
  - b) attributes correspond to tracked columns of tracked table,
  - c) an attribute *HS\_Hist* of *HS\_History* type.

## 6.2.2 HS\_HistoryTable Method

### Purpose

Obtain a whole history table.

### Definition

```
CREATE STATIC METHOD HS_HistoryTable()
  RETURNS TABLE(<AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History)
  FOR <TableTypeIdentifier>
  BEGIN
    RETURN TABLE(
      SELECT <AllPKeyColumns>, <AllTrackedColumns>, HS_Hist
      FROM <TableIdentifier>;
    )
  END
```

### Description

- 1) Definition of this method uses the following substitutes.
  - a) <AllPKeyColumns>: Comma-separated primary key column names of a tracked table.
  - b) <AllTrackedColumns>: Comma-separated tracked column names of a tracked table.
  - c) <AllPKeyColumnsDef>: Comma-separated character strings which concatenate the primary key column name of a tracked table and the name of data type of the column.
  - d) <AllTrackedColumnsDef>: Comma-separated character strings which concatenate the tracked column name of a tracked table and the name of data type of the column.
  - e) <TableTypeIdentifier>: The value of the result of invoking *HS\_ConstructTableTypeIdentifier* with the name of a tracked table as its argument.
  - f) <TableIdentifier>: The value of the result of invoking *HS\_ConstructTableIdentifier* with the name of a tracked table as its argument.
- 2) The *HS\_HistoryTable()* method returns whole history table. If a history table is generated as a base table, the *HS\_HistoryTable()* method selects all columns from the history table with no condition, and return it.

## 6.2.3 HS\_PNormalize Methods

## Purpose

Obtain a period-normalized table of the specified columns of a history table.

## Definition

```

CREATE STATIC METHOD HS_PNormalize
  (targetColumn CHARACTER VARYING(<ColumnNameLength>))
  RETURNS TABLE(<AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History)
  FOR <TableTypeIdentifier>
  BEGIN
    RETURN HS_PNormalize(ARRAY[targetColumn]);
  END

--
-- This method shall be created
-- if and only if there is only one tracked column.
--

CREATE STATIC METHOD HS_PNormalize
  (targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
  RETURNS TABLE(<AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History)
  FOR <TableTypeIdentifier>
  BEGIN
    RETURN HS_HistoryTable();
  END

--
-- This method shall be created
-- if and only if there are more than one tracked columns.
--

CREATE STATIC METHOD HS_PNormalize
  (targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY)
  RETURNS TABLE(<AllPKeyColumnsDef>, <AllTrackedColumnsDef>, HS_Hist HS_History)
  FOR <TableTypeIdentifier>
  BEGIN
    DECLARE m_tableTypeIdentifier CHARACTER VARYING(<IdentifierLength>);
    DECLARE m_periodColumnsComparison_HT1_HT2 CLOB;
    DECLARE m_periodColumns CLOB;
    DECLARE m_periodColumns_HT1 CLOB;
    DECLARE m_periodColumns_HT2 CLOB;
    DECLARE m_tmpAllColumns CLOB;
    DECLARE m_tmpAllColumns_HT1 CLOB;
    DECLARE m_tmpAllPKeyColumns CLOB;
    DECLARE m_tmpPKeyComparison_HT1_HT2 CLOB;
    DECLARE m_stmt CLOB;
    DECLARE m_setStmt CLOB;
    DECLARE m_Result_Tbl
      ROW(<AllPKeyColumnsDef>, <AllTrackedColumnsDef>,
        HS_Hist HS_History) MULTISSET;
    DECLARE m_i INTEGER;
    DECLARE m_restColumns CLOB;
    DECLARE m_dlmString CHAR(2) DEFAULT ',';
    DECLARE m_empString CHAR VARYING(1) DEFAULT '';
    DECLARE m_endPosition INTEGER;
    DECLARE m_columnName CLOB;
    DECLARE m_tmpTargetColumns
      CHARACTER VARYING(<ColumnNameLength>) ARRAY;

    SET m_tableTypeIdentifier = '<TableTypeIdentifier>';

```

```

SET m_tmpAllColumns = '<AllPKeyColumns>, <AllTrackedColumns>';
SET m_tmpAllColumns_HT1 = '<AllPKeyColumns_HT1>, <AllTrackedColumns_HT1>';
SET m_tmpAllPKeyColumns = '<AllPKeyColumns>';
SET m_tmpPKeyComparison_HT1_HT2 = '<AllPKeyComparison_HT1_HT2>';
SET m_periodColumns = '';
SET m_periodColumns_HT1 = '';
SET m_periodColumns_HT2 = '';

SET m_i = 1;
WHILE m_i <= CARDINALITY(targetColumns) DO
    SET m_tmpTargetColumns[m_i] =
        HS_ConstructColumnIdentifier(targetColumns[m_i]);
    SET m_i = m_i + 1;
END WHILE;
SET m_i = 1;
WHILE m_i <= CARDINALITY(targetColumns) DO
    IF m_i > 1 THEN
        SET m_periodColumns_HT1 = m_periodColumns_HT1 || ', ' ;
        SET m_periodColumns_HT2 = m_periodColumns_HT2 || ', ' ;
    END IF;
    SET m_periodColumns_HT1 = m_periodColumns_HT1 ||
        'HT1.' || targetColumns[m_i] ;
    SET m_periodColumns_HT2 = m_periodColumns_HT2 ||
        'HT2.' || targetColumns[m_i] ;
    SET m_i = m_i + 1;
END WHILE;
SET m_periodColumnsComparison_HT1_HT2 =
    '(' || m_periodColumns_HT1 || ')' ||
    'IS NOT DISTINCT FROM (' || m_periodColumns_HT2 || ')' ;
SET m_restColumns = m_tmpAllColumns || m_dlmString;
WHILE m_restColumns <> m_empString DO
    SET m_endPosition = POSITION(m_dlmString IN m_restColumns) - 1;
    SET m_columnName =
        SUBSTRING(m_restColumns FROM 1 FOR m_endPosition);
    IF POSITION(m_columnName || m_dlmString
        IN m_tmpAllPKeyColumns || m_dlmString) >= 1 THEN
        SET m_periodColumns = m_periodColumns || m_columnName || ', ';
    ELSEIF NOT EXISTS(SELECT *
        FROM UNNEST(m_tmpTargetColumns) AS CLMS(CNAME)
        WHERE CNAME = m_columnName) THEN
        SET m_periodColumns = m_periodColumns ||
            'NULLIF(' || m_columnName || ', ' || m_columnName || ')' || ', ';
    ELSE
        SET m_periodColumns = m_periodColumns || m_columnName || ', ';
    END IF;
    SET m_restColumns =
        OVERLAY(m_restColumns PLACING m_empString
            FROM 1 FOR m_endPosition + CHAR_LENGTH(m_dlmString));
END WHILE;
SET m_stmt =
    'WITH RECURSIVE PeriodTemp(' || m_tmpAllColumns || ', HS_Hist) AS' ||
    '(' ||
    ' SELECT ' || m_periodColumns || 'HS_Hist' ||
    ' FROM TABLE(' ||
    '     m_tableTypeIdentifier || '::HS_HistoryTable()) AS HT' ||
    ' UNION ALL' ||
    ' SELECT ' || m_tmpAllColumns_HT1 || ', ' ||
    '     NEW HS_History(HT1.HS_Hist.HS_BeginTime,' ||
    '     HT2.HS_Hist.HS_EndTime)' ||
    ' FROM PeriodTemp HT1,' ||
    '     TABLE(' ||

```



- c) *<AllPrimaryKeyColumns\_HT1>*: Comma-separated primary key column names with prefix 'HT1' of a tracked table. For every primary key column of a tracked table, the column name appears exactly once.
- d) *<AllTrackedColumns\_HT1>*: Comma-separated tracked column names with prefix 'HT1' of a tracked table. For every tracked column of a tracked table, the column name appears exactly once.
- e) *<AllPrimaryKeyColumnsDef>*: Comma-separated character strings of column definitions of primary key columns of a tracked table. Each character string forms a pair of a column name and the data type of the column with intervening a space. For every primary key column of a tracked table, the column name appears exactly once.
- f) *<AllTrackedColumnsDef>*: Comma-separated character strings of column definitions of tracked columns of a tracked table. Each character string forms a pair of a column name and the data type of the column with intervening a space. For every tracked column of a tracked table, the column name appears exactly once.
- g) *<AllPrimaryKeyComparison\_HT1\_HT2>*: Equality comparison predicates in order to test whether the values of primary key column with qualifier HT1 and that with qualifier HT2 are equal for all primary key columns. For every primary key column of a tracked table, the predicate appears exactly once.
- h) *<TableTypeIdentifier>*: The value of the result of invoking *HS\_ConstructTableTypeIdentifier* with the name of a tracked table as its argument.
- 2) The *HS\_PNormalize(CHARACTER VARYING(<ColumnNameLength>))* method takes the following input parameter:
- a) *targetColumn*, 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.
- 3) The *HS\_PNormalize(CHARACTER VARYING(<ColumnNameLength>) ARRAY)* method takes the following input parameter:
- a) *targetColumns*, 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.
- 4) The *HS\_PNormalize(CHARACTER VARYING(<ColumnNameLength>))* method returns the result which is returned as the result of the following method invocation:
- HS\_PNormalize(ARRAY[targetColumn])*
- 5) The *HS\_PNormalize(CHARACTER VARYING(<ColumnNameLength>) ARRAY)* method returns a period-normalized table of specified columns of a history table.

## 7 SQL/MM History Information Schema

### 7.1 Introduction

The SQL/MM History Information Schema views are defined as being in a schema named *HS\_INFORMTN\_SCHEMA* enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views is granted to PUBLIC WITH GRANT OPTION so that they can be queried by any user and so that SELECT privilege can be further granted on views that reference these Information Schema views.

In order to provide access to the same information that is available via the *HS\_INFORMTN\_SCHEMA* to an SQL-Agent in an SQL-environment where the SQL-implementation does not support Feature F391, "Long identifiers" in Part 2 of ISO/IEC 9075, alternative views are provided that use only short identifiers.

An implementation may define objects that are associated with *HS\_INFORMTN\_SCHEMA* that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

## 7.2 HS\_TRACKED\_TABLES view

### Purpose

List the tables that have associated history table.

### Definition

```
CREATE VIEW HS_TRACKED_TABLES AS
  SELECT DISTINCT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
  FROM HS_DEFINITION_SCHEMA.HS_TRACKED_TABLES
```

### 7.3 HS\_TRACKED\_COLUMNS view

#### Purpose

Identify the tracked columns of all tracked tables.

#### Definition

```
CREATE VIEW HS_TRACKED_COLUMNS AS
  SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
  FROM HS_DEFINITION_SCHEMA.HS_TRACKED_COLUMNS
```

## 8 SQL/MM History Definition Schema

### 8.1 Introduction

The only purpose of the SQL/MM History Definition Schema is to provide a data model to support the *HS\_INFORMTN\_SCHEMA* and to assist understanding. The base tables of the SQL/MM History Definition Schema are defined as being in a schema named *HS\_DEFINITION\_SCHEMA*. These base tables are effectively updated when *HS\_CreateHistory* procedure is invoked. The table definitions are as complete as the definitional power of ISO/IEC 9075 allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1. The function of the definition is stated.
2. The SQL definition of the object is presented as a <table definition>.
3. An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

## 8.2 HS\_TRACKED\_TABLES base table

### Purpose

List the tracked tables.

### Definition

```
CREATE TABLE HS_TRACKED_TABLES
(
  TABLE_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
  TABLE_SCHEMA INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
  TABLE_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,

  CONSTRAINT HS_TRACKED_TABLES_PRIMARY_KEY
    PRIMARY KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME),
  CONSTRAINT TABLE_EXISTS
    FOREIGN KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME)
    REFERENCES DEFINITION_SCHEMA.TABLES
      (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME)
)
```

### Description

- 1) The values of *TABLE\_CATALOG*, *TABLE\_SCHEMA*, and *TABLE\_NAME* are the fully qualified name of the tracked table.

### 8.3 HS\_TRACKED\_COLUMNS base table

#### Purpose

List the tracked columns of the tracked tables.

#### Definition

```
CREATE TABLE HS_TRACKED_COLUMNS
(
  TABLE_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
  TABLE_SCHEMA INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
  TABLE_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,
  COLUMN_NAME INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,

  CONSTRAINT HS_TRACKED_COLUMNS_PRIMARY_KEY
    PRIMARY KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME),
  CONSTRAINT COLUMN_EXISTS
    FOREIGN KEY(TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
    REFERENCES DEFINITION_SCHEMA.COLUMNS
      (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME)
)
```

#### Description

- 1) The values of *TABLE\_CATALOG*, *TABLE\_SCHEMA*, and *TABLE\_NAME* are the catalog name, the unqualified schema name, and the qualified identifier, respectively, of the table containing the column being described.
- 2) The values of *COLUMN\_NAME* is the name of the tracked column being specified.

## 9 Status Codes

The character string value returned in an SQLSTATE parameter comprises a 2-character class value followed by a 3-character subclass value. The class value for each condition and the subclass value or values for each class value are specified in Table 4 – SQLSTATE class and subclass values.

The "Category" column has the following meanings: "S" means that the class value given corresponds to successful completion and is a completion condition; "W" means that the class value given corresponds to a successful completion but with a warning and is a completion condition; "N" means that the class value corresponds to a no-data situation and is a completion condition; "X" means that the class value given corresponds to an exception condition.

For a successful completion code but with a warning, the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class value for *warning* (defined in Subclause 23.1, "SQLSTATE" in ISO/IEC 9075).

For an exception completion code, the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class value *SQL routine exception* (defined in Subclause 23.1, "SQLSTATE" in Part 2 of ISO/IEC 9075-2).

**Table 4 – SQLSTATE class and subclass values**

Category	Condition	Class	Subcondition	Subclass
X	SQL/MM History exception	2F	table name is a null value	F01
X	SQL/MM History exception	2F	table does not exist	F02
X	SQL/MM History exception	2F	history table already exists	F03
X	SQL/MM History exception	2F	no tracked column is specified	F04
X	SQL/MM History exception	2F	column name is a null value	F05
X	SQL/MM History exception	2F	column does not exist	F06
X	SQL/MM History exception	2F	history table does not exist	F07
X	SQL/MM History exception	2F	table already exists	F08
X	SQL/MM History exception	2F	beginning of period is a null value	F09
X	SQL/MM History exception	2F	end of period precedes beginning of period	F10
X	SQL/MM History exception	2F	no overlap exist	F11
X	SQL/MM History exception	2F	failed to generate a single continuous period	F12
X	SQL/MM History exception	2F	both end points of period are null	F15
X	SQL/MM History exception	2F	empty period	F16
X	SQL/MM History exception	2F	tracked table has no primary key column	F17
X	SQL/MM History exception	2F	a primary key column cannot be specified as a tracked column explicitly	F18
X	SQL/MM History exception	2F	result of the intersect operation is empty period	F19
X	SQL/MM History exception	2F	the two periods do not meet or overlap	F20
X	SQL/MM History exception	2F	result of the except operation has disjoint periods	F21
X	SQL/MM History exception	2F	result of the except operation is empty period	F22
X	SQL/MM History exception	2F	beginning of period succeeds current timestamp	F23
X	SQL/MM History exception	2F	end of period succeeds current timestamp	F24
X	SQL/MM History exception	2F	a given timestamp value expresses future time	F25



## 10 Conformance

### 10.1 Requirements for conformance

A conforming implementation shall support all of the following public user-defined types and routines:

- 1) *HS\_CreateHistory*(*CHARACTER VARYING*(<TableNameLength>), *CHARACTER VARYING*(<ColumnNameLength>) *ARRAY*) procedure, which defines <TableTypeIdentifier> type,
- 2) *HS\_DropHistory*(*CHARACTER VARYING*(<TableNameLength>)) procedure,
- 3) *HS\_History* type,
- 4) All of methods of *HS\_History* type which are defined in Subclause 6.1, "HS\_History Type and Routines" in this part of ISO/IEC 13249.

A conforming implementation shall support the views comprising the History Information Schema as defined in Clause 7, "SQL/MM History Information Schema".

### 10.2 Features of ISO/IEC 9075 required in this part of ISO/IEC 13249

This part of ISO/IEC 13249 requires the following features defined in ISO/IEC 9075.

- Feature S024, "Enhanced structured types"
- Feature T326, "Table functions"
- Feature F052, "Intervals and datetime arithmetic"
- Feature S091, "Basic array support"
- Feature S096, "Optional array bounds"
- Feature S201, "SQL-invoked routines on arrays"

This part of ISO/IEC 13249 requires either of the following features defined in ISO/IEC 9075.

- Feature T651, "SQL-schema statements in SQL routines"
- Feature T653, "SQL-schema statements in external routines"

This part of ISO/IEC 13249 requires either of the following features defined in ISO/IEC 9075.

- Feature T652, "SQL-dynamic statements in SQL routines"
- Feature T654, "SQL-dynamic statements in external routines"

This part of ISO/IEC 13249 requires the following feature defined in ISO/IEC 9075 if the length of <qualified identifier> of a tracked table is greater than 10 characters, where <qualified identifier> is a syntactic element (BNF nonterminal symbol) defined in ISO/IEC 9075.

- Feature F391, "Long identifiers"

### 10.3 Claims of conformance

Claims of conformance to this part of ISO/IEC 13249 shall state:

- 1) The definitions for all elements and actions that this part of ISO/IEC 13249 specifies as implementation-defined.

## Annex A

(informative)

### Example Application

#### A.1 Introduction

In order to help understanding of this part of ISO/IEC 13249, hereafter are example applications.

Assume an SQL-database that holds information on teachers of a certain university. In this SQL-database, there is a table "emp" that has the following structure shown in Figure 1.

EmpID	EmpName	Title	Salary	Dept	DateOfBirth
1	Tom	Assistant	4000	CS1	1970-01-01

**Figure 1 Example Table "emp"**

Definition of the table "emp" is as shown below.

```
CREATE TABLE emp(
  EmpID INTEGER NOT NULL,
  EmpName CHARACTER VARYING(32),
  Title CHARACTER VARYING(32),
  Salary INTEGER,
  Dept CHARACTER VARYING(32),
  DateOfBirth DATE,
  PRIMARY KEY(EmpID)
)
```

#### A.2 Storing History Rows

Information stored in the table "emp" will be updated as Salary and/or Dept are changed. The following operations to the table "emp" is assumed to be executed in order.

Seq. No	Begin time	Operation
(1)	1996/04/01 15:23:52	Invoke the procedure HS_CreateHistory('emp', ARRAY['EmpName', 'Title', 'Salary', 'Dept'])
(2)	1997/04/02 10:34:19	Insert a row { 2, 'Ken', 'Assistant Professor', 7000, 'Med2', '1960-05-03' }.
(3)	1998/04/02 09:47:52	Update the following column in the row which the value of EmpID is 2: Title: 'Assistant Professor' --> 'Professor' Salary: 7000 --> 8000
(4)	1999/04/02 13:06:23	Update the following column in the row which the value of EmpID is 1: Salary: 4000 --> 5000
(5)	2000/04/03 11:12:28	Update the following column in the row which the value of EmpID is 1: Title: 'Assistant' --> 'Assistant Professor' Salary: 5000 --> 6000
(6)	2001/04/02 10:04:46	Update the following column in the row which the value of EmpID is 2: Dept: 'Med2' --> 'Med1'
(7)	2002/04/02 12:17:03	Update the following column in the row which the value of EmpID is 2: Dept: 'Med1' --> 'Med3'
(8)	2003/04/02 09:58:11	Update the following column in the row which the value of EmpID is 1: Dept: 'CS1' --> 'CS2'
(9)	2004/04/02 11:47:33	Delete row which the value of EmpID is 1.

The following are detailed explanations on processing that will be automatically executed when the above processing are made.

- 1) In order to start storing history, invoke procedure of HS\_CreateHistory('emp', ARRAY['EmpName', 'Title', 'Salary', 'Dept']) at 1996/04/01 15:23:52.

By this, the following processing are automatically executed.

a) Automatic generation of the following:

- i) The structured type HS\_TYPE\_emp associated with the history table for table "emp"

HS\_TYPE\_emp has the following attributes.

- Attributes corresponding to the primary key columns in the table "emp."
- Attributes corresponding to tracked columns of the table "emp."
- Attribute HS\_Hist, whose declared type is structured type HS\_History composed of a begin time and an end time.

- ii) history table corresponding to the table "emp"

Table returned by the HS\_TYPE\_emp::HS\_HistoryTable method is typed by HS\_TYPE\_emp.

- iii) HS\_TYPE\_emp provides the following methods.

- 1) HS\_PNormalize(targetColumns CHARACTER VARYING(<ColumnNameLength>) ARRAY):

Returns a table that is the result of period normalization of selected column(s), which is specified with the input parameter "targetColumns", of a history table.

2) HS\_HistoryTable():

Returns the whole history table.

- iv) An insert trigger to maintain the history table after an insert operation to the table "emp" is executed.
- v) An update trigger to maintain the history table after an update operation to the table "emp" is executed.
- vi) A delete trigger to maintain the history table after a delete operation to the table "emp" is executed.

b) Add value of HS\_Hist column (NEW HS\_History(CURRENT\_TIMESTAMP, CAST(NULL AS TIMESTAMP) )) to every row included in the table "emp" and insert it into the history table.

At this point, the history table for the table "emp" becomes as shown below:

EmpID	EmpName	Title	Salary	Dept	HS_Hist (HS_BeginTime, HS_EndTime)
1	Tom	Assistant	4000	CS1	(1996-04-01 15:23:52, NULL)

2) Insert a new row { 2, 'Ken', 'Assistant Professor', 7000, 'Med2', '1960-05-03' } into the table "emp" at 1997/04/02 10:34:19.

An insert trigger for the table "emp" is activated on this insert operation, and the following processing is executed:

a) A new history row { 2, 'Ken', 'Assistant Professor', 7000, 'Med2', NEW HS\_History (TIMESTAMP'1997-04-02 10:34:19', CAST(NULL AS TIMESTAMP)) } is inserted into the history table.

At this point, the history table for the table "emp" becomes as shown below:

EmpID	EmpName	Title	Salary	Dept	HS_Hist (HS_BeginTime, HS_EndTime)
1	Tom	Assistant	4000	CS1	(1996-04-01 15:23:52, NULL)
2	Ken	Assistant Professor	7000	Med2	(1997-04-02 10:34:19, NULL)

3) At 1998/04/02 09:47:52, update Title of the row whose EmpID is 2 from 'Assistant Professor' to 'Professor' and also Salary from 7000 to 8000.

An update trigger for the table "emp" is activated on this update operation, and the following processing is executed:

a) The end time of the history row, which was inserted into the history table in 2)-a) is set to 1998-04-02 09:47:52.

b) A new history row { 2, 'Ken', 'Professor', 8000, 'Med2', NEW HS\_History (TIMESTAMP'1998-04-02 09:47:52', CAST(NULL AS TIMESTAMP) ) } is inserted into the history table.

At this point, the history table for the table "emp" becomes as shown below:

EmpID	EmpName	Title	Salary	Dept	HS_Hist (HS_BeginTime, HS_EndTime)
1	Tom	Assistant	4000	CS1	(1996-04-01 15:23:52, NULL)
2	Ken	Assistant Professor	7000	Med2	(1997-04-02 10:34:19, 1998-04-02 09:47:52)
2	Ken	Professor	8000	Med2	(1998-04-02 09:47:52, NULL)

- 4) At 1999/04/02 13:06:23, update Salary of the row whose EmpID is 1 from 4000 to 5000.
- 5) At 2000/04/03 11:12:28, update Title of the row whose EmpID is 1 from 'Assistant' to 'Assistant Professor' and also Salary from 5000 to 6000.
- 6) At 2001/04/02 10:04:46, update Dept of the row whose EmpID is 2 from 'Med2' to 'Med1'.
- 7) At 2002/04/02 12:17:03, update Dept of the row whose EmpID is 2 from 'Med1' to 'Med3'.
- 8) At 2003/04/02 09:58:11, update Dept of the row whose EmpID is 1 from 'CS1' to 'CS2'.

At this point, the history table for the table "emp" becomes as shown below:

EmpID	EmpName	Title	Salary	Dept	HS_Hist (HS_BeginTime, HS_EndTime)
1	Tom	Assistant	4000	CS1	(1996-04-01 15:23:52, 1999-04-02 13:06:23)
1	Tom	Assistant	5000	CS1	(1999-04-02 13:06:23, 2000-04-03 11:12:28)
1	Tom	Assistant Professor	6000	CS1	(2000-04-03 11:12:28, 2003-04-02 09:58:11)
1	Tom	Assistant Professor	6000	CS2	(2003-04-02 09:58:11, NULL)
2	Ken	Assistant Professor	7000	Med2	(1997-04-02 10:34:19, 1998-04-02 09:47:52)
2	Ken	Professor	8000	Med2	(1998-04-02 09:47:52, 2001-04-02 10:04:46)
2	Ken	Professor	8000	Med1	(2001-04-02 10:04:46, 2002-04-02 12:17:03)
2	Ken	Professor	8000	Med3	(2002-04-02 12:17:03, NULL)

- 9) At 2004/04/02 11:47:33, delete the row whose EmpID is 1.

A delete trigger for the table "emp" is activated on this delete operation, and the following processing is executed:

- a) The end time of the history row, which was inserted into the history table in 6)-e) is set to 2004-04-02 11:47:33.

At this point, the history table for the table "emp" becomes as shown below:

EmpID	EmpName	Title	Salary	Dept	HS_Hist (HS_BeginTime, HS_EndTime)
1	Tom	Assistant	4000	CS1	(1996-04-01 15:23:52, 1999-04-02 13:06:23)
1	Tom	Assistant	5000	CS1	(1999-04-02 13:06:23, 2000-04-03 11:12:28)
1	Tom	Assistant Professor	6000	CS1	(2000-04-03 11:12:28, 2003-04-02 09:58:11)
1	Tom	Assistant Professor	6000	CS2	(2003-04-02 09:58:11, 2004-04-02 11:47:33)
2	Ken	Assistant Professor	7000	Med2	(1997-04-02 10:34:19, 1998-04-02 09:47:52)
2	Ken	Professor	8000	Med2	(1998-04-02 09:47:52, 2001-04-02 10:04:46)
2	Ken	Professor	8000	Med1	(2001-04-02 10:04:46, 2002-04-02 12:17:03)
2	Ken	Professor	8000	Med3	(2002-04-02 12:17:03, NULL)

### A.3 Example of Queries to History Table

The example of the search to the history table generated by the processing shown in the foregoing paragraph is described below.

In addition, the method of HS\_History type currently used in the example of search has the following meanings.

- HS\_Contains(TIMESTAMP): Tests whether the period which a certain HS\_History value represents contains the specified TIMESTAMP value.
- HS\_Overlaps(TIMESTAMP, TIMESTAMP): Tests whether the period which a certain HS\_History value represents overlaps with the period which the specified two TIMESTAMP values represent.
- HS\_Meets(TIMESTAMP): Tests whether the end time of the period which a certain HS\_History value represents is equal to the specified TIMESTAMP value;
- HS\_MonthInterval(): Obtain the length of the period which a certain HS\_History value represents as year-month interval.

NOTE 7 – The virtualization approach that uses a method instead of a direct table reference is adopted in this part of ISO/IEC 13249.

1) How much is the salary of each employee at 2001/07/01 00:00:00?

```
SELECT EmpName, Salary, HS_Hist.HS_BeginTime, HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT
WHERE HS_Hist.HS_Contains(TIMESTAMP'2001-07-01 00:00:00') = 1
```

Answer:

EmpName	Salary	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Tom	6000	2000-04-01 00:00:00	2003-04-01 00:00:00
Ken	8000	2001-04-01 00:00:00	2002-04-01 00:00:00

2) Who was an assistant professor at 2001/10/01 00:00:00?

```
SELECT EmpName, Title, HS_Hist.HS_BeginTime, HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT
WHERE HS_Hist.HS_Contains(TIMESTAMP'2001-10-01 00:00:00') = 1
AND Title = 'Assistant Professor'
```

Answer:

EmpName	Title	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Tom	Assistant Professor	2000-04-01 00:00:00	2003-04-01 00:00:00

3) Whose salary was the highest at 2002/01/01 00:00:00?

```
SELECT EmpName, Salary, HS_Hist.HS_BeginTime, HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT
WHERE HS_Hist.HS_Contains(TIMESTAMP'2002-01-01 00:00:00') = 1
AND Salary = (
SELECT MAX(Salary)
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT2
WHERE HS_Hist.HS_Contains(TIMESTAMP'2002-01-01 00:00:00') = 1)
```

Answer:

EmpName	Salary	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Ken	8000	2001-04-01 00:00:00	2002-04-01 00:00:00

4) Who was an assistant professor from 2000/05/01 00:00:00 to 2003/02/01 00:00:00?

```
SELECT EmpName, Title, HS_Hist.HS_BeginTime, HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT
WHERE HS_Hist.HS_Overlaps(
TIMESTAMP'2000-05-01 00:00:00',
TIMESTAMP'2003-02-01 00:00:00') = 1
AND Title = 'Assistant Professor'
```

Answer:

EmpName	Title	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Tom	Assistant Professor	2000-04-01 00:00:00	2003-04-01 00:00:00

5) How much is the present salary of the person who was an assistant professor at 1997/10/01 00:00:00 and was a professor at 2001/05/01 00:00:00?

```
SELECT HT3.EmpName, HT3.Salary, HT1.Title,
HT1.HS_Hist.HS_BeginTime, HT1.HS_Hist.HS_EndTime,
HT2.Title,
HT2.HS_Hist.HS_BeginTime, HT2.HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT1,
TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT2,
TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT3
WHERE HT1.HS_Hist.HS_Contains(TIMESTAMP'1997-10-01 00:00:00') = 1
AND HT1.Title = 'Assistant Professor'
AND HT2.HS_Hist.HS_Contains(TIMESTAMP'2001-05-01 00:00:00') = 1
AND HT2.Title = 'Professor'
AND HT3.HS_Hist.HS_Meets(CURRENT_TIMESTAMP) = 1
AND HT1.EmpID = HT2.EmpID
AND HT1.EmpID = HT3.EmpID
```

Answer:

EmpName	Salary	Title	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime	Title	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Ken	8000	Assistant Professor	1997-04-01 00:00:00	1998-04-01 00:00:00	Professor	2001-04-01 00:00:00	2002-04-01 00:00:00

6) Who belonged to the Med2 Department continuously for over two years?

```
SELECT HT.EmpName, HT.Dept,
       HT.HS_Hist.HS_BeginTime, HT.HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_PNormalize('Dept')) AS HT
WHERE HT.Dept = 'Med2'
      AND HT.HS_Hist.HS_MonthInterval() >= INTERVAL '2' YEAR
```

Answer:

EmpName	Dept	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Ken	Med2	1997-04-01 00:00:00	2001-04-01 00:00:00

7) Who belonged to the Med2 Department for over two years in total?

```
SELECT HT.EmpName, HT.Dept, HT.Intv
FROM (
  SELECT EmpID, EmpName, Dept,
         SUM(HS_Hist.HS_MonthInterval()) AS Intv
  FROM TABLE(HS_TYPE_emp::HS_PNormalize('Dept')) AS HT2
  GROUP BY EmpID, EmpName, Dept) AS HT
WHERE HT.Intv YEAR >= INTERVAL '2' YEAR
      AND HT.Dept = 'Med2'
```

Answer:

EmpName	Dept	Intv
Ken	Med2	'4' YEAR

8) Who has the longest period as an assistant professor?

```
SELECT HT1.EmpName, HT1.Title,
       HT1.HS_Hist.HS_BeginTime, HT1.HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_PNormalize('Title')) AS HT1
WHERE HT1.Title = 'Assistant Professor'
      AND HT1.HS_Hist.HS_MonthInterval() = (
  SELECT MAX(HT2.HS_Hist.HS_MonthInterval())
  FROM TABLE(HS_TYPE_emp::HS_PNormalize('Title')) AS HT2
  WHERE HT2.Title = 'Assistant Professor')
```

Answer:

EmpName	Title	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Tom	Assistant Professor	2000-04-01 00:00:00	2004-04-01 00:00:00

9) About each employee, how much was the salary at the time when the employee became a professor?

```
SELECT HT1.EmpName, HT1.Title, HT1.Salary,
```

```

HT1.HS_Hist.HS_BeginTime, HT1.HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT1
WHERE HT1.Title = 'Professor'
AND HT1.HS_Hist.HS_BeginTime = (
  SELECT MIN(HS_Hist.HS_BeginTime)
  FROM TABLE(HS_TYPE_emp::HS_HistoryTable()) AS HT2
  WHERE EmpID = HT1.EmpID AND Title = 'Professor')

```

Answer:

EmpName	Title		HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Ken	Professor	8000	1998-04-01 00:00:00	2001-04-01 00:00:00

10) Who belonged to the CS1 Department continuously for over three years?

```

SELECT EmpName, Dept,
       HS_Hist.HS_BeginTime, HS_Hist.HS_EndTime
FROM TABLE(HS_TYPE_emp::HS_PNormalize('Dept')) AS HT
WHERE HT.Dept = 'CS1'
AND HT.HS_Hist.HS_MonthInterval() >= INTERVAL '3' YEAR

```

Answer:

EmpName	Dept	HS_Hist.HS_BeginTime	HS_Hist.HS_EndTime
Tom	CS1	1996-04-01 15:23:52	2003-04-01 00:00:00

## Bibliography

- [1] ISO/IEC 9075-3:2008, *Information technology — Database languages — SQL — Part 3:Call-Level Interface (SQL/CLI)*.
- [2] ISO/IEC 9075-9:2008, *Information technology — Database languages — SQL — Part 9:Management of External Data (SQL/MED)*.
- [3] ISO/IEC 9075-10:2008, *Information technology — Database languages — SQL — Part 10:Object Language Bindings (SQL/OLB)*.
- [4] ISO/IEC 9075-13:2008, *Information technology — Database languages — SQL — Part 13:SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)*.
- [5] ISO/IEC 9075-14:2008, *Information technology — Database languages — SQL — Part 14:XML-Related Specifications (SQL/XML)*.