# ISO/IEC JTC 1/SC 32 N 1306

Date: 2005-05-04

REPLACES: --

**ISO/IEC JTC 1/SC 32**

**Data Management and Interchange**

**Secretariat: United States of America (ANSI)**
**Administered by Farance Inc. on behalf of ANSI**

| | |
|---|---|
| **DOCUMENT TYPE** | Working Draft Text |
| **TITLE** | Working Draft for NP Ballot 32N1305 – SQL Multimedia and Application Packages – Part 7: History |
| **SOURCE** | JTC 1/SC 32/WG 3 |
| **PROJECT NUMBER** | 1.32.04.03.07.00 |
| **STATUS** | This is a Working Draft for the NP Ballot in 32N1305 |
| **REFERENCES** | |
| **ACTION ID.** | FYI |
| **REQUESTED ACTION** | |
| **DUE DATE** | |
| **Number of Pages** | 92 |
| **LANGUAGE USED** | English |
| **DISTRIBUTION** | P & L Members<br>SC Chair<br>WG Conveners and Secretaries |

Douglas Mann, Secretary, ISO/IEC JTC 1/SC 32
Farance Inc *, 360 Pelissier Lake Road, Marquette, MI 49855-9678, United States of America
Telephone: +1 906-249-9275; E-mail: MannD@battelle.org
available from the JTC 1/SC 32 WebSite http://jtc1sc32.org/
*Farance Inc. administers the ISO/IEC JTC 1/SC 32 Secretariat on behalf of ANSI

# ISO/IEC JTC 1/SC 32/WG 4 N0047

## TXL-013r1

Date: 2005-04-11

**ISO/IEC WD 13249-7**

ISO/IEC JTC 1/SC 32/WG 4

Secretariat: xxxx

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

*Technologies de I'information – Languages de bases de donées — Multimédia SQL et paquetages d'application — Partie 7: Historie*

**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: (20) Preparatory
Document language: E

C:\Documents and Settings\mannd.MILKY-WAY\My Documents\A-Mann\AA-Mann\DM-sc32\Docs-32\Incomming\Berlin-docs\32N1306-WG4N0047-txl013r1-Part-7-History-candidate-base-document.doc STD Version 2.0

blank page

# Contents

Page

# Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 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, *JTC*, Subcommittee SC 32, *SC*.

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*

# 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, "Conformance", defines the criteria for conformance to this part of ISO/IEC 13249.

3) Clause 3, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 13249, constitute provisions of this part of ISO/IEC 13249.

4) Clause 4, "Definitions, notations, and conventions", defines the notations and conventions used in this part of ISO/IEC 13249.

5) Clause 5, "Concepts", presents concepts used in the definition of this part of ISO/IEC 13249.

6) Clause 6, "History Procedures", defines the history associated routines.

7) Clause 7, "History Types", defines the user-defined types provided for the manipulation of history.

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

9) Clause 9, "SQL/MM History Definition Schema" defines the SQL/MM History Definition Schema.

10) Clause 10, "Status Codes", defines the SQLSTATE codes used in 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

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 the history routines and history user-defined types.

The history routines defined in this part of ISO/IEC 13249 adhere to the following:

— A history routine is generic to history handling. It addresses the need to store, manage, and retrieve information based on aspects of history data such as values of changed objects before and after each change for insert, update, and delete for tables.

An implementation of this part of ISO/IEC 13249 may exist in environments that also support information and content management, decision support, data mining, and data warehousing systems.

Application areas addressed by implementations of this part of ISO/IEC 13249 include, but are not restricted to, personnel information management, geoengineering, multimedia, scientific research, and resource management applications.

# 2 Conformance

## 2.1 Requirements for conformance

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

## 2.3 Claims of conformance

# 3 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.

## 3.1 ISO/IEC JTC 1 standards

ISO/IEC 9075-1:2003, *Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework)*.

ISO/IEC 9075-2:2003, *Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation)*.

ISO/IEC 9075-4:2003, *Information technology - Database languages - SQL - Part 4: Persistent Stored Modules (SQL/PSM)*.

ISO/IEC 9075-11:2003, *Information technology - Database languages - SQL - Part 11: Information and Definition Schemas (SQL/Schemata)*.

ISO/IEC 13249-1:2002, *Information Technology - Database Languages - SQL Multimedia and Application Packages - Part 1: Framework*.

# 4 Terms and definitions

## 4.1 Definitions

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

### 4.1.1 Definitions provided in Part 1

This part of ISO/IEC 13249 makes use of all terms defined in Part 1 of ISO/IEC 13249.

### 4.1.2 Definitions provided in Part 7

This part of ISO/IEC 13249 defines the following terms:

**4.1.2.1**
**current state row**
A row in a current state table.

**4.1.2.2**
**current state table**
A base table attended with a history table.

**4.1.2.3**
**history begin time**
An attribute of user-defined type HS_History in a history table, its name is HS_HistoryBeginTime, its data type is TIMESTAMP, and its value records CURRENT_TIMESTAMP of the row inserted or updated in the current state table.

**4.1.2.4**
**history end time**
An attribute of user-defined type HS_History in a history table, its name is HS_HistoryEndTime, its data type is TIMESTAMP, and its value records CURRENT_TIMESTAMP of the row updated or deleted in the current state table.

**4.1.2.5**
**history row**
A row in a history table.

**4.1.2.6**
**history row set**
A set of history rows, which have the same value of primary key of a current state table.

**4.1.2.7**
**history table**
A table that holds change history for the current state table. The history table consists of all columns in the current state table and the column of structured type, which has attributes of history begin time and history end time.

When a history table is created, all rows in the current state table are inserted with the history begin time set to the CURRENT_TIMESTAMP.

When Insert operation to the current state table is executed, a row in the history table is inserted. When Update operation to the current state table is executed, history end time of the latest row in the history row set is updated, and a row in the history table is inserted. When Delete operation to the current state table is executed, history end time of the latest row in the history row set is updated.

## 4.2 Notations

### 4.2.1 Notations provided in Part 1

The notations used in this part of ISO/IEC 13249 are defined in ISO/IEC 13249-1:2002.

### 4.2.2 Notations provided in Part 7

This part of ISO/IEC 13249 uses the prefix 'HS_' for user-defined type, attribute and SQL-invoked routine names.

## 4.3 Conventions

The conventions used in this part of ISO/IEC 13249 are defined in ISO/IEC 13249-1:2002

# 5 Concepts

## 5.1 Overview

When a change operation is made for some row in a table in an SQL database, the value held just before the change operation is lost. Even if you need the value before change after the change operation, you cannot have the value before change anymore.

In Part 7, when you specify to hold change history for a base table in an SQL database, a history table is generated for each base table. History table is a table that holds change history for the current state table. History table consists of all columns in the current state table and the column of structured type, which has attributes of history begin time and history end time.

When Insert operation to the original base table is executed, a row in the history table is inserted. When Update operation to the original base table is executed, history end time of the latest row in the history row set is updated to the CURRENT_TIMESTAMP value, and a row in the history table is inserted. When Delete operation to the original base table is executed, history end time of the latest row in the history row set is updated to the CURRENT_TIMESTAMP value.

When a user wants to specify the value of history begin time or history end time of the latest history row in some history row set clearly, after a history row is generated, the value of history begin time or history end time is updated by invoking the method provided in Part 7.

The original base table of the history table is to store only latest values among change history stored in the history table, therefore, we named the original table of history table “current state table.” Also, we named specific history row selected by some value of the primary key in the current state table “current state row.”

The processing to add a history row to a history table is automatically executed by defining triggers that fires when Update/Insert/Delete is executed, to the current state table.

## 5.2 Example Application

In order to help understanding of part 7, hereafter are example applications.

Assume an SQL database that holds information on teachers of a certain university. In this database, a table “emp” having the following structure shown in Fig. 1 exists.

| EmpID | EmpName | Title | Salary | Dept |
|---|---|---|---|---|
| 1 | Tom | Assistant | 4000 | CS1 |

**Figure 1 Example Table "emp"**

Definitions of the table “emp” are as shown below.

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

HS_History type is pre-provided in order to show period that a history row has. HS_History type has two attributes to hold history begin time and history end time. Also, HS_History type has definitions of some methods to judge cross relations between a period and another that HS_History value has.

### 5.2.1 Storing History Rows

Information stored in the table “emp” will be updated as Salary or Dept changes. Also, by updating values of Salary or Dept, the history begin time value of the history row, which was inserted to the history table HS_TBL_emp, became machine clock time at the time of the processing, therefore, update the value to the date of change that has logical meaning. Concretely speaking, execute the following processing to the table “emp” in order.

| Machine clock time | Operation | Changed date<br>(the value of history begin time) |
|---|---|---|
| 1996/04/01 15:23:52 | Call the procedure HS_CreateHistory('emp', ARRAY['Title', 'Salary', 'Dept']) | |
| 1997/04/02 10:34:19 | Insert a row { 2, ‘Ken’, ‘Assistant Professor’, 7000, ‘Med2’ }. | 1997/04/01 00:00:00 |
| 1998/04/02 09:47:52 | Update the following column in the row which the value of EmpID is 2:<br>Title: 'Assistant Professor' --> 'Professor'<br>Salary: 7000 --> 8000 | 1998/04/01 00:00:00 |
| 1999/04/02 13:06:23 | Update the following column in the row which the value of EmpID is 1:<br>Salary: 4000 --> 5000 | 1999/04/01 00:00:00 |
| 2000/04/03 11:12:28 | Update the following column in the row which the value of EmpID is 1:<br>Title: 'Assistant' --> 'Assistant Professor'<br>Salary: 5000 --> 6000 | 2000/04/01 00:00:00 |
| 2001/04/02 10:04:46 | Update the following column in the row which the value of EmpID is 2:<br>Dept: 'Med2' --> 'Med1' | 2001/04/01 00:00:00 |
| 2002/04/02 12:17:03 | Update the following column in the row which the value of EmpID is 2:<br>Dept: 'Med1' --> 'Med3' | 2002/04/01 00:00:00 |
| 2003/04/02 09:58:11 | Update the following column in the row which the value of EmpID is 1:<br>Dept: 'CS1' --> 'CS2' | 2003/04/01 00:00:00 |
| 2004/04/02 11:47:33 | Delete row which the value of EmpID is 1. | 2004/04/01 00:00:00 |

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, call procedure and execute HS_CreateHistory('emp', ARRAY['Title', 'Salary', 'Dept']) to 1996/04/01 15:23:52.

   By this, the following processing are automatically executed.

   a) Automatical generation of the following:

      i) HS_TYPE_emp to show the structure of history table for table “emp”

      HS_TYPE_emp has the following attributes.

      - Attribute corresponding to every column in the table “emp”

- HS_History value HS_Hist to store history begin time and history end time.

ii) history table HS_TBL_emp corresponding to the table “emp”

Table HS_TBL_emp is typed by HS_TYPE_emp.

iii) HS_TYPE_emp provides the following methods.

1) HS_Period(TargetColumns information_schema.SQL_IDENTIFIER ARRAY):

Table with the same structure as history table corresponding to “emp” is returned.

The table that is returnedby the HS_Period method is a table that was re-structured to hold only the changes of column values specified with input parameter ”TargetColumns.” For the columns that were not specified with the input parameter, stored is the value of the start time of the period among the values of Column HS_Hist of each row of the table that returned by the HS_Period method.

2) HS_HistoryBeginTime(HS_PRM_EmpID INTEGER, HS_BeginOfPeriod TIMESTAMP):

Among history rows included in the history table corresponding to the table ”emp,” specify history begin time of the latest history row in the history row set corresponding to the primary key.

3) HS_HistoryEndTime(HS_PRM_EmpID INTEGER, HS_EndOfPeriod TIMESTAMP):

Among history rows included in the history table corresponding to the table ”emp,” specify history end time of the latest history row in the history row set corresponding to the primary key.

iv) Trigger to insert the row after the update to the history table as a history row when Update is executed for the table “emp.”

v) Trigger to insert the row to the history table as a history row when Insert is executed for the table “emp.”

vi) Trigger to specify CURRENT_TIMESTAMP to history end time of the latest history row corresponding to the deleted current state row when delete is executed to the table “emp.”

Two triggers to specify prohibition of updates except latest history row and timestamp in the history table in order to keep consistency within the history table when update is executed to the history table corresponding to the table “emp.”

b) Add value of HS_Hist column (NEW HS_History( CURRENT_TIMESTAMP, NULL )) to every row included in the table “emp” and insert it to 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_HistoryBeginTime | HS_HistoryEndTime |
| 1 | Tom | Assistant | 4000 | CS1 | 1996-04-01 15:23:52 | NULL |

2) Insert a new row { 2, ‘Ken’, ‘Assistant Professor’, 7000, ‘Med2’ } to the table “emp” for 1997/04/02 10:34:19.

Insert trigger for the table “emp” is ignited by this insert, 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', NULL) } is inserted to 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_HistoryBeginTime | HS_HistoryEndTime |
| 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) In order to change history begin time value to which machine clock time 1997-04-02 10:34:19 is specified, to 1997-04-01 00:00:00 having logical meaning such as date of order, execute HS_TYPE_emp::HS_HistoryBeginTime( 2, TIMESTAMP'1997-04-01 00:00:00' ) for 1997/04/02 10:34:19.

   By this processing, history begin time of the history row, which was inserted to the history table in 2) is changed from 1997-04-02 10:34:19 to 1997-04-01 00:00:00.

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

| EmpID | EmpName | Title | Salary | Dept | HS_Hist | |
|---|---|---|---|---|---|---|
| | | | | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 1 | Tom | Assistant | 4000 | CS1 | 1996-04-01 15:23:52 | NULL |
| 2 | Ken | Assistant Professor | 7000 | Med2 | 1997-04-01 00:00:00 | NULL |

4) For 1998/04/02 09:47:52, update Title of the row whose EmpID is 2 from ‘Assistant’ to ‘Assistant Professor’ and also Salary from 5000 to 6000.

   By this update, update trigger for the table “emp” is ignited by this update, and the following processing is executed:

   a) History end time of the history row , whish was inserted to the history table in 2)-a) is specified 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', NULL ) } is inserted to the history table.

5) For 1998/04/02 09:47:52, execute HS_TYPE_emp::HS_HistoryBeginTime ( 2, TIMESTAMP'1998-04-01 00:00:00' ) in the same way as 3).

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

| EmpID | EmpName | Title | Salary | Dept | HS_Hist | |
|---|---|---|---|---|---|---|
| | | | | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 1 | Tom | Assistant | 4000 | CS1 | 1996-04-01 15:23:52 | NULL |
| 2 | Ken | Assistant Professor | 7000 | Med2 | 1997-04-01 00:00:00 | 1998-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med2 | 1998-04-01 00:00:00 | NULL |

6) .Make the following updates in the same way as 4) and 5).

   a) For 1999/04/02 13:06:23 and 1999/04/02 13:07:34, update Salary of the row whose EmpID is 1 from 4000 to 5000, and execute HS_TYPE_emp::HS_HistoryBeginTime ( 1, TIMESTAMP'1999-04-01 00:00:00' ) in the same way as 4) and 5) respectively.

b) For 2000/04/03 11:12:28 and 2000/04/03 11:13:39, update Title of the row whose EmpID is 1 from 'Assistant' to 'Assistant Professor' and also Salary from 5000 to 6000, and execute HS_TYPE_emp::HS_HistoryBeginTime ( 1, TIMESTAMP'2000-04-01 00:00:00' ) in the same way as 4) and 5) respectively

c) For 2001/04/02 10:04:46 and 2001/04/02 10:05:57, update Dept of the row whose EmpID is 2 from 'Med2' to 'Med1' and execute HS_TYPE_emp::HS_HistoryBeginTime( 2, TIMESTAMP'2001-04-01 00:00:00' ) in the same way as 4) and 5) respectively.

d) For 2002/04/02 12:17:03 and 2002/04/02 12:18:14, update Dept of the row whose EmpID is 2 from 'Med1' to 'Med3' and execute HS_TYPE_emp::HS_HistoryBeginTime( 2, TIMESTAMP'2002-04-01 00:00:00' ) in the same way as 4) and 5) respectively.

e) For 2003/04/02 09:58:11 and 2003/04/02 09:59:22, update Dept of the row whose EmpID is 1 from 'CS1' to 'CS2' and execute HS_TYPE_emp::HS_HistoryBeginTime( 1, TIMESTAMP'2003-04-01 00:00:00' ) in the same way as 4) and 5) respectively.

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

| EmpID | EmpName | Title | Salary | Dept | HS_Hist | |
|---|---|---|---|---|---|---|
| | | | | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 1 | Tom | Assistant | 4000 | CS1 | 1996-04-01 15:23:52 | 1999-04-01 00:00:00 |
| 1 | Tom | Assistant | 5000 | CS1 | 1999-04-01 00:00:00 | 2000-04-01 00:00:00 |
| 1 | Tom | Assistant Professor | 6000 | CS1 | 2000-04-01 00:00:00 | 2003-04-01 00:00:00 |
| 1 | Tom | Assistant Professor | 6000 | CS2 | 2003-04-01 00:00:00 | NULL |
| 2 | Ken | Assistant Professor | 7000 | Med2 | 1997-04-01 00:00:00 | 1998-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med2 | 1998-04-01 00:00:00 | 2001-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med1 | 2001-04-01 00:00:00 | 2002-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med3 | 2002-04-01 00:00:00 | NULL |

7) For 2004/04/02 11:47:33, delete the row whose EmpID is 1.

By this delete, delete trigger for the table "emp" is ignited by this delete, and the following processing is executed:

a) History end time of the history row, whish was inserted to the history table in 6)-e) is specified to 2004-04-02 11:47:33.

8) For 2004/04/02 11:48:44, execute HS_TYPE_emp::HS_HistoryEndTime ( 1, TIMESTAMP'2004-04-01 00:00:00' ).

By this processing, the following processing is executed:

a) History end time of the history row, whish was inserted to the history table in 6)-e) is changed from 2004-04-02 11:47:33 to 2004-04-01 00:00.

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

| EmpID | EmpName | Title | Salary | Dept | HS_Hist | |
|---|---|---|---|---|---|---|
| | | | | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 1 | Tom | Assistant | 4000 | CS1 | 1996-04-01 15:23:52 | 1999-04-01 00:00:00 |
| 1 | Tom | Assistant | 5000 | CS1 | 1999-04-01 00:00:00 | 2000-04-01 00:00:00 |
| 1 | Tom | Assistant Professor | 6000 | CS1 | 2000-04-01 00:00:00 | 2003-04-01 00:00:00 |
| 1 | Tom | Assistant Professor | 6000 | CS2 | 2003-04-01 00:00:00 | 2004-04-01 00:00:00 |
| 2 | Ken | Assistant Professor | 7000 | Med2 | 1997-04-01 00:00:00 | 1998-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med2 | 1998-04-01 00:00:00 | 2001-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med1 | 2001-04-01 00:00:00 | 2002-04-01 00:00:00 |
| 2 | Ken | Professor | 8000 | Med3 | 2002-04-01 00:00:00 | NULL |

### 5.2.2 Searching for History Table

The example of the search to history table HS_TBL_emp 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 expresses contains the specified TIMESTAMP value.
- HS_Overlaps(TIMESTAMP, TIMESTAMP): Tests whether the period which a certain HS_History value expresses overlaps with the period which two specified TIMESTAMP values expresses.
- HS_Meets(TIMESTAMP): Tests whether the end time of the period which a certain HS_History value expresses is equal to the specified TIMESTAMP value;
- HS_MonthInterval(): Obtain length of the period which a certain HS_History value expresses as year-month time interval.

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

```
SELECT EmpName, Salary, HS_Hist
   FROM HS_TBL_emp
   WHERE HS_Hist.HS_Contains(TIMESTAMP'2001-07-01 00:00:00')
```

Answer:

| EmpName | Salary | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 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 on 2001/10/01 00:00:00?

```
SELECT EmpName, Title, HS_Hist
   FROM HS_TBL_emp
   WHERE HS_Hist.HS_Contains(TIMESTAMP'2001-10-01 00:00:00')
      AND Title = 'Assistant Professor'
```

Answer:

| EmpName | Title | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| Tom | Assistant Professor | 2000-04-01 00:00:00 | 2003-04-01 00:00:00 |

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

```
SELECT EmpName, Salary, HS_Hist
   FROM HS_TBL_emp
   WHERE HS_Hist.HS_Contains(TIMESTAMP'2002-01-01 00:00:00')
      AND Salary = (
         SELECT MAX(Salary)
         FROM HS_TBL_emp
         WHERE HS_Hist.HS_Contains(TIMESTAMP'2002-01-01 00:00:00'))
```

Answer:

| EmpName | Salary | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 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
   FROM HS_TBL_emp
   WHERE HS_Hist.HS_Overlaps(
         TIMESTAMP'2000-05-01 00:00:00',
         TIMESTAMP'2003-02-01 00:00:00')
      AND Title = 'Assistant Professor'
```

Answer:

| EmpName | Title | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 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 on 1997/10/01 00:00:00 and was an professor on 2001/05/01 00:00:00?

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

Answer:

| EmpName | Salary | Title | HS_Hist | | Title | HS_Hist | |
|---|---|---|---|---|---|---|---|
| | | | HS_HistoryBeginTime | HS_HistoryEndTime | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 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 2 years?

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

Answer:

| EmpName | Dept | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| Ken | Med2 | 1997-04-01 00:00:00 | 2001-04-01 00:00:00 |

7) Who belonged to the Med2 Department for over 2 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_Period('Dept'))
         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
   FROM TABLE(HS_TYPE_emp::HS_Period('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_Period('Title')) AS HT2
            WHERE HT2.Title = 'Assistant Professor')
```

Answer:

| EmpName | Title | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 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
   FROM HS_TBL_emp HT1
   WHERE HT1.Title = 'Professor'
      AND HT1.HS_Hist.HS_HistoryBeginTime = (
         SELECT MIN(HS_Hist.HS_HistoryBeginTime)
         FROM HS_TBL_emp
         WHERE EmpID = HT1.EmpID AND Title = 'Professor')
```

Answer:

| EmpName | Title | Salary | HS_Hist | |
|---|---|---|---|---|
| | | | HS_HistoryBeginTime | HS_HistoryEndTime |
| 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 3 years?

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

Answer:

| EmpName | Dept | HS_Hist | |
|---|---|---|---|
| | | HS_HistoryBeginTime | HS_HistoryEndTime |
| Tom | CS1 | 1996-04-01 15:23:52 | 2003-04-01 00:00:00 |

## 5.3 Structure of History Table

Corresponding to the update/insert operation to some current state row, a row is inserted to a history table. In part 7, the row inserted to a history table is called a history row. Furthermore, if the update/insert operation is performed to some current state row, a history row corresponding to the current state row is inserted to the history table. Such history rows can be grouped as rows corresponding to a certain current state row. In part 7, a collection of history row corresponding to a certain specific current state row is called a history row set of the current state row.

In part 7, the following type is provided to express attributes of a history row:

- The HS_History type with the following attributes:
  - HS_HistoryBeginTime: a TIMESTAMP value to store history begin time;
  - HS_HistoryEndTime: a TIMESTAMP value to store history end time;

The *HS_HistoryBeginTime* attribute which *HS_History* type has is set as the value of CURRENT_TIMESTAMP in the time of the history row being generated. When a user wants to specify the value of *HS_HistoryBeginTime* attribute clearly, after a history row is generated, the value of *HS_HistoryBeginTime* attribute is set up by the invocation of the *HS_HistoryStartTime* method of the *HS_TYPE_<TableName>* type. However, only when corresponding history row is the newest history row in history row set, the value of HS_HistoryBeginTime attribute can be changed.

The following adjustments must be maintained about history rows in the history table:

- The time order of the value of history begin time which a history row contained in a certain history row set has must be the same as an order that a history row was added to a history table.

- History begin time of a history row added to a history table by insert/update operation to current state table must be equal to history end time of a history row added immediately before in the same history row set.

A history row is expressed by the following type which is automatically generated for every current state table:

- The *HS_TYPE_<TableName>* type with the following attributes:

  - attributes correspond to all columns of the current state table;

  - HS_Hist: a *HS_History* value to store the period which history row expresses;

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

We provide the *HS_CreateHistory* procedure to generate a history table corresponding to the specified table (current state table).

In order to store in a history table automatically the value of the past which a current state table held and to maintain the history table, the approach which used the trigger is adopted in part 7. The following triggers are defined:

- A trigger to insert a history row to the history table when value of the specified columns of corresponding current state table is updated.

- A trigger to insert a history row to the history table when a row is inserted to the corresponding current state table.

- A trigger to update history end time of the latest history row to the CURRENT_TIMESTAMP value when a row is deleted from the corresponding current state table.

- triggers to execute the following operation when value of the HS_Hist column of the history table is updating or updated:

  - forbid updating other than history row of the latest in a certain history row set.

  - maintain the consistency of the values of history begin time and history end time.

These triggers are automatically generated for every current state table when the *HS_CreateHistory* procedure is called.

## 5.5 Referring History Table

To refer the history table, you should specify the name of the history table *HS_TBL_<TableName>* directly in the query, or use the return value of the *HS_Period* method of *HS_TYPE_<TableName>* type.

### 5.6 Concepts taken from ISO/IEC 9075

## 5.7 Types representing history rows

### 5.7.1 HS_History type

#### 5.7.1.1 Attributes of the HS_History type

The *HS_History* type is an abstraction for attributes of a history row, using the following attributes:

- *HS_HistoryBeginTime*: The begin time of the period which has the corresponding history row;
- *HS_HistoryEndTime*: The end time of the period which has the corresponding history row;

#### 5.7.1.2 Methods of the HS_History type

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

- *HS_History*: Generate *HS_History* value which has the specified TIMESTAMP values as the history begin time and the history end time;
- *HS_Overlaps*: Tests whether the period which a certain *HS_History* value expresses overlaps with the period which two specified TIMESTAMP values expresses;
- *HS_Overlaps*: Tests whether the period which a certain *HS_History* value expresses overlaps with the period which a *HS_History* value expresses;
- *HS_Meets*: Tests whether the end time of the period which a certain *HS_History* value expresses is equal to the specified TIMESTAMP value;
- *HS_Meets*: Tests whether the end time of the period which a certain *HS_History* value expresses is equal to the begin time of the period which a *HS_History* value expresses;
- *HS_Precedes*: Tests whether the whole period which a certain *HS_History* value expresses precedes the specified TIMESTAMP value;
- *HS_Precedes*: Tests whether the whole period which a certain *HS_History* value expresses precedes the begin time of the period which a *HS_History* value expresses;
- *HS_Contains*: Tests whether the period which a certain *HS_History* value expresses contains the specified TIMESTAMP value;
- *HS_Contains*: Tests whether the period which a certain *HS_History* value expresses contains the period which two specified TIMESTAMP value expresses;
- *HS_Contains*: Tests whether the period which a certain *HS_History* value expresses contains the whole period which a *HS_History* value expresses;
- *HS_Equals*: Tests whether the period which a certain *HS_History* value expresses is equal to the period which two specified TIMESTAMP value expresses;
- *HS_Equals*: Tests whether the period which a certain *HS_History* value expresses is equal to the period which a *HS_History* value expresses;
- *HS_MonthInterval*: Obtain length of the period which a certain *HS_History* value expresses as year-month time interval;
- *HS_DayInterval*: Obtain length of the period which a certain *HS_History* value expresses as day-time time interval;

- *HS_Intersect*: Generate a new *HS_History* value with period which is overlap of the period which a certain *HS_History* value expresses and the period which two specified TIMESTAMP value expresses;

- *HS_Intersect*: Generate a new *HS_History* value with period which is overlap of the period which a certain *HS_History* value expresses and the period which the specified *HS_History* value expresses;

### 5.7.2 HS_TYPE_<TableName> type

#### 5.7.2.1 Attributes of the HS_TYPE_<TableName> type

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

- attributes correspond to columns of a current state table;
- a HS_History value HS_Hist;

#### 5.7.2.2 Methods of the HS_TYPE_<TableName> type

The type *HS_TYPE_<TableName>* provides the following methods for public use:

- HS_HistoryBeginTime: Set the value of history begin time of the latest history row in the specified history row set;
- HS_HistoryEndTime: Set the value of history end time of the latest history row in the specified history row set;
- HS_Period: Obtain restructured history table which stores change history of the specified column;
- HS_Period: Obtain restructured history table which stores change history of the specified columns;

## 5.8 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 is listed in Table 1 – Method and function name correspondences. Since the names of these functions are unique, their parameter types are not given.

**Table 1 – Method and function name correspondences**

| Type Name | Method Name | Parameter Types (if any) | Function Name |
|---|---|---|---|
| HS_History | HS_History | TIMESTAMP,<br>TIMESTAMP | HS_History |
| HS_History | HS_Overlaps | TIMESTAMP,<br>TIMESTAMP | HS_Overlaps |
| HS_History | HS_Overlaps | HS_History | HS_Overlaps |
| HS_History | HS_Meets | TIMESTAMP | HS_Meets |
| HS_History | HS_Meets | HS_History | HS_Meets |
| HS_History | HS_Precedes | TIMESTAMP | HS_Precedes |
| HS_History | HS_Precedes | HS_History | HS_Precedes |
| HS_History | HS_Contains | TIMESTAMP | HS_Contains |
| HS_History | HS_Contains | TIMESTAMP,<br>TIMESTAMP | HS_Contains |
| HS_History | HS_Contains | HS_History | HS_Contains |
| HS_History | HS_Equals | TIMESTAMP,<br>TIMESTAMP | HS_Equals |
| HS_History | HS_Equals | HS_History | HS_Equals |
| HS_History | HS_MonthInterval | | HS_MonthInterval |
| HS_History | HS_DayInterval | | HS_DayInterval |
| HS_History | HS_Intersect | TIMESTAMP,<br>TIMESTAMP | HS_Intersect |
| HS_History | HS_Intersect | HS_History | HS_Intersect |
| HS_TYPE_<TableName> | HS_Period | information_schema.SQL_IDENTIFIER | HS_Period |
| HS_TYPE_<TableName> | HS_Period | information_schema.SQL_IDENTIFIER ARRAY | HS_Period |
| HS_TYPE_<TableName> | HS_HistoryBeginTime | <AllPKeyColumnsDef>,<br>TIMESTAMP | HS_HistoryBeginTime |
| HS_TYPE_<TableName> | HS_HistoryEndTime | <AllPKeyColumnsDef>,<br>TIMESTAMP | HS_HistoryEndTime |

## 5.9 The History Information Schema

# 6 History Procedures

The procedures in this section provide for beginning to store history rows and suspending to store history rows.

## 6.1 HS_CreateHistory Procedures and Sub Procedures

### 6.1.1 HS_CreateHistory Procedures

**Purpose**

The HS_CreateHistory procedure calls some sub-procedures. These sub-procedures generate a history table for the specified current state table, generate five triggers to maintain the history table, define the type correspond to the specified current state table, initialize the history table, and so on.

**Definition**

```
CREATE PROCEDURE HS_CreateHistory(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN HistoryColumns information_schema.SQL_IDENTIFIER ARRAY
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      CALL HS_CreateHistoryErrorCheck(TableName, HistoryColumns);
      CALL HS_CreateHistoryTableType(TableName);
      CALL HS_CreateHistoryTable(TableName);
      CALL HS_CreateUpdateTrigger(TableName, HistoryColumns);
      CALL HS_CreateInsertTrigger(TableName);
      CALL HS_CreateDeleteTrigger(TableName);
      CALL HS_CreateHTForbidUpdateHistoryRowsTrigger(TableName);
      CALL HS_CreateHTConsistencyMaintenanceTrigger(TableName);
      CALL HS_CreateHistoryTimeMethod(TableName);
      CALL HS_CreatePeriodMethod(TableName);
      CALL HS_InitializeHistoryTable(TableName);
   END
```

**Definitional Rules**

1) If error occurred in one of sub-procedures called in the HS_CreateHistory procedure, rollback of all operation should be executed.

**Description**

1) The *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) an information_schema.SQL_IDENTIFIER ARRAY value *HistoryColumns*.

2) The *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* calls the following procedures in order:

   a) the *HS_CreateHistoryErrorCheck* procedure,

   b) the *HS_CreateHistoryTableType* procedure,

   c) the *HS_CreateHistoryTable* procedure,

   d) the *HS_CreateUpdateTrigger* procedure,

e) the *HS_CreateInsertTrigger* procedure,

f) the *HS_CreateDeleteTrigger* procedure,

g) the *HS_CreateHTForbidUpdateHistoryRowsTrigger* procedure,

g+1) the *HS_CreateHTConsistencyMaintenanceTrigger* procedure,

h) the *HS_CreateHistoryTimeMethod* procedure,

i) the *HS_CreatePeriodMethod* procedure,

j) the *HS_InitializeHistoryTable* procedure.

***** Editor's Note *****

Possible Problem

The names of history table, triggers and type generated in HS_CreateHistory procedure may exceed 18characters.

***** Editor's Note *****

Possible Problem

The schema structure of current state table currently assumed in the trigger created by execution of *HS_CreateHistory* procedure may be changed with ALTER TABLE operations to current state table.
When a certain column is added to current state table, the change history of the value of the added column is not held. Moreover, when a certain column is deleted from current state table, the column not existing may be referred to in a trigger.
In first edition, the ALTER TABLE operation to current state table shall not be assumed. When performing ALTER TABLE operation to current state table, adjustment shall be held on a user's responsibility.
In the future, if it comes to be able to perform history managements also including information schema, it will be thought that it is more useful.

***** Editor's Note *****

Possible Problem

When the foreign key is defined to current state table, the problem how a foreign key should be defined in history table can be considered.
Since it is a thing showing what state history table suited when current state table had the past, when the target of a foreign key is changed, history table should not be changed. Therefore, it is thought that a foreign key should not be set to history table.

***** Editor's Note *****

Possible Problem

A problem occurs when an error is in the value in history row accumulated in history table. When the values in a history row accumulated in history table contain error, the result of the query to the history table contains error. Therefore, as for the error in history table, it is desirable that it can be corrected.
From a viewpoint of the falsification of history table, as for the value in history row once inserted in history table, it is desirable to prevent from changing.
In this edition, we make it impossible to do updating to history table. However, there is room of discussion about the propriety of updating the value in the history table.

### 6.1.2 HS_CreateHistoryErrorCheck Procedure

**Purpose**

Checks whether the input parameters of HS_CreateHistory procedure is valid.

**Definition**

```
CREATE PROCEDURE HS_CreateHistoryErrorCheck(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN HistoryColumns information_schema.SQL_IDENTIFIER ARRAY
   )
   LANGUAGE SQL
   DETERMINISTIC
   READS SQL DATA
   BEGIN
      IF TableName IS NULL THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: TableName is null.';
      END IF;
      IF NOT EXISTS(SELECT *
            FROM information_schema.tables
            WHERE table_name = TableName) THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: Table '||TableName||' does not exist.';
      END IF;
      IF EXISTS(SELECT *
            FROM information_schema.tables
            WHERE table_name = 'HS_TBL_'||TableName) THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: History table HS_TBL_'||
            TableName||' already exists.';
      END IF;
      IF HistoryColumns IS NULL THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: HistoryColumns is null.';
      END IF;
      IF CARDINALITY(HistoryColumns) < 1 THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: No history column is specified.';
      END IF;
      SET i = 1;
      WHILE i <= CARDINALITY(HistoryColumns) DO
         IF HistoryColumns[i] IS NULL THEN
            SIGNAL XXXXX SET MESSAGE_TEXT =
               'Illegal input: Element of HistoryColumns has null value.';
         ELSEIF NOT EXISTS(SELECT *
               FROM information_schema.columns
               WHERE table_name = TableName
                  AND column_name = HistoryColumns[i]) THEN
            SIGNAL XXXXX SET MESSAGE_TEXT =
               'Illegal input: Column '||HistoryColumns[i]||
               ' does not exist in table '||TableName||'.';
         END IF;
         SET i = i + 1;
      END WHILE;
   END
```

**Definitional Rules**

1) The *HS_CreateHistoryErrorCheck(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure should be called only in the

*HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateHistoryErrorCheck(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) an information_schema.SQL_IDENTIFIER ARRAY value *HistoryColumns*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

3) The value passed as the second parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *HistoryColumns*.

4) For the procedure *HS_CreateHistoryErrorCheck(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)*:

   a) Case:

      i) If *TableName* is the null value, then an exception condition is raised: SQL/MM History exception – XXXXX.

      ii) If *TableName* is table name which does not exist in the database, then an exception condition is raised: SQL/MM History exception – XXXXX.

      iii) If *HS_TBL_<TableName>* is table name which exists in the database, then an exception condition is raised: SQL/MM History exception – XXXXX.

      iv) If *HistoryColumns* is the null value, then an exception condition is raised: SQL/MM History exception – XXXXX.

      v) If *HistoryColumns* has no element, then an exception condition is raised: SQL/MM History exception – XXXXX.

      vi) If ether of element of *HistoryColumns* is the null value or column name which does not exist in the table specified by *TableName*, then an exception condition is raised: SQL/MM History exception – XXXXX.

      vii) Otherwise, the procedure is completed normally.

### 6.1.3 HS_CreateHistoryTableType Procedure

**Purpose**

Generates the *HS_TYPE_<TableName>* type, which is the base type of a history table corresponding to the specified current state table.

**Definition**

```
CREATE PROCEDURE HS_CreateHistoryTableType(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumnsDef CLOB;
      DECLARE tmpAllPKeyColumnsDef_HS_PRM CLOB;

      CALL HS_CreateCommaSeparatedAllColumnAndTypeList(
         TableName, NULL, NULL, tmpAllColumnsDef);
      CALL HS_CreateCommaSeparatedPrimaryKeyAndTypeList(
         TableName, 'HS_PRM_', NULL, tmpAllPKeyColumnsDef_HS_PRM);

      SET stmt = 'CREATE TYPE HS_TYPE_'||TableName||' AS('||
         tmpAllColumnsDef||', HS_Hist HS_History)'||

         'STATIC METHOD HS_Period('||
         '  TargetColumn information_schema.SQL_IDENTIFIER)'||
         ' RETURNS TABLE('||tmpAllColumnsDef||', HS_Hist HS_History),'||

         'STATIC METHOD HS_Period('||
         '  TargetColumns information_schema.SQL_IDENTIFIER ARRAY)'||
         ' RETURNS TABLE('||tmpAllColumnsDef||', HS_Hist HS_History),'||

         'STATIC METHOD HS_HistoryBeginTime('||
         tmpAllPKeyColumnsDef_HS_PRM||', HS_BeginOfPeriod TIMESTAMP)'||
         ' RETURNS TIMESTAMP,'||

         'STATIC METHOD HS_HistoryEndTime('||
         tmpAllPKeyColumnsDef_HS_PRM||', HS_EndOfPeriod TIMESTAMP)'||
         ' RETURNS TIMESTAMP';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateHistoryTableType(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateHistoryTableType(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

### 6.1.4 HS_CreateHistoryTable Procedure

**Purpose**

Generates a history table corresponding to the specified current state table.

**Definition**

```
CREATE PROCEDURE HS_CreateHistoryTable(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllPKeyColumns CLOB;
      DECLARE tmpAllPKeyColumnsOpt CLOB;

      CALL HS_CreateCommaSeparatedPrimaryKeyList(
         TableName, NULL, NULL, tmpAllPKeyColumns);
      CALL HS_CreateCommaSeparatedPrimaryKeyList(
         TableName, NULL, ' WITH OPTIONS NOT NULL',
         tmpAllPKeyColumnsOpt);

      SET stmt = 'CREATE TABLE HS_TBL_'||TableName||
         ' OF HS_TYPE_'||TableName||
         '(REF IS HS_REF_'||TableName||' SYSTEM GENERATED,'||
         tmpAllPKeyColumnsOpt||','||
         'PRIMARY KEY ('||tmpAllPKeyColumns||
         ', HS_Hist.HS_HistoryBeginTime)'||')';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateHistoryTable(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

2) For the generated history table:

   a) Privilege of deleting rows from the history table is not granted to anyone.

   b) Privilege of inserting rows to the history table is granted only to the triggers generated in the HS_CreateHistory procedure, and is not granted to the others.

   c) Privilege of updating the HS_Hist column is granted to everyone, and privilege of updating the other columns is not granted to everyone.

**Description**

1) The *HS_CreateHistoryTable(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

### 6.1.5 HS_CreateUpdateTrigger Procedure

**Purpose**

Generates a trigger to insert a history row to the history table when value of the specified columns which correspond to current state table is updated.

**Definition**

```
CREATE PROCEDURE HS_CreateUpdateTrigger(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN HistoryColumns information_schema.SQL_IDENTIFIER ARRAY
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllColumns_nwr CLOB;
      DECLARE tmpAllPKeyComparison_nwr CLOB;
      DECLARE tmpAllHistoryColumns CLOB;
      DECLARE tmpAllHistoryColumnsComparison CLOB;

      DECLARE i INTEGER;

      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, NULL, NULL, tmpAllColumns);
      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, 'nwr.', NULL, tmpAllColumns_nwr);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, NULL, 'nwr.', tmpAllPKeyComparison_nwr);

      SET i = 1;
      SET tmpAllHistoryColumns = '';
      SET tmpAllHistoryColumnsComparison = '';
      WHILE i <= CARDINALITY(HistoryColumns) DO
         IF i > 1 THEN
            SET tmpAllHistoryColumns = tmpAllHistoryColumns||',';
            SET tmpAllHistoryColumnsComparison =
               tmpAllHistoryColumnsComparison||' AND ';
         END IF
         SET tmpAllHistoryColumns =
            tmpAllHistoryColumns||HistoryColumns[i];
         SET tmpAllHistoryColumnsComparison =
            tmpAllHistoryColumnsComparison||
            'odr.'||HistoryColumns[i]||' = '||'nwr.'||HistoryColumns[i];

         SET i = i + 1;
      END WHILE;

      SET stmt = stmt||
         'CREATE TRIGGER HS_TR_'||TableName||'_UPD'||
         ' AFTER UPDATE OF '||tmpAllHistoryColumns||' ON '||TableName||
         ' REFERENCING NEW ROW AS nwr'||
         '             OLD ROW AS odr'||
         ' FOR EACH ROW'||
         ' WHEN(NOT('||tmpAllHistoryColumnsComparison||'))'||
         ' BEGIN ATOMIC';
      SET stmt = stmt||
         'UPDATE HS_TBL_'||TableName||
```

```
        ' SET HS_Hist.HS_HistoryEndTime = CURRENT_TIMESTAMP'||
        ' WHERE '||tmpAllPKeyComparison_nwr||
        ' AND HS_Hist.HS_HistoryBeginTime ='||
        ' (SELECT MAX( HS_Hist.HS_HistoryBeginTime )'||
        ' FROM HS_TBL_'||TableName||
        ' WHERE '||tmpAllPKeyComparison_nwr||
        ' AND HS_Hist.HS_HistoryBeginTime <'||
        ' CURRENT_TIMESTAMP);';
    SET stmt = stmt||
        ' INSERT INTO HS_TBL_'||TableName||'('||
        tmpAllColumns||', HS_Hist'||
        ') VALUES ('||
        tmpAllColumns_nwr||', '||
        'NEW HS_History(CURRENT_TIMESTAMP, CAST(NULL AS TIMESTAMP)));';
    SET stmt = stmt||' END';

    EXECUTE IMMEDIATE stmt;
END
```

**Definitional Rules**

1) The *HS_CreateUpdateTrigger(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateUpdateTrigger(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) an information_schema.SQL_IDENTIFIER ARRAY value *HistoryColumns*.

2) The value passed as the first parameter of the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

3) The value passed as the second parameter of the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *HistoryColumns*.

### 6.1.6 HS_CreateInsertTrigger Procedure

**Purpose**

Generates a trigger to insert a history row to the history table when a row is inserted to the corresponding current state table.

**Definition**

```
CREATE PROCEDURE HS_CreateInsertTrigger(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllColumns_nwr CLOB;

      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, NULL, NULL, tmpAllColumns);
      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, 'nwr.', NULL, tmpAllColumns_nwr);

      SET stmt = 'CREATE TRIGGER HS_TR_'||TableName||'_INS'||
         ' AFTER INSERT ON '||TableName||
         ' REFERENCING NEW ROW AS nwr'||
         ' FOR EACH ROW'||
         ' BEGIN ATOMIC';

      SET stmt = stmt||
         ' INSERT INTO HS_TBL_'||TableName||' ( '||
         tmpAllColumns||', HS_Hist'||
         ') VALUES ('||
         tmpAllColumns_nwr||', '||
         'NEW HS_History(CURRENT_TIMESTAMP, CAST(NULL AS TIMESTAMP)));';

      SET stmt = stmt||' END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateInsertTrigger(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateInsertTrigger(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

### 6.1.7 HS_CreateDeleteTrigger Procedure

**Purpose**

Generates a trigger to update history end time of latest history row to the CURRENT_TIMESTAMP value when a row is deleted from the corresponding current state table.

**Definition**

```
CREATE PROCEDURE HS_CreateDeleteTrigger(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllPKeyComparison_odr CLOB;

      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, NULL, 'odr.', tmpAllPKeyComparison_odr);

      SET stmt = 'CREATE TRIGGER HS_TR_'||TableName||'_DEL'||
         ' AFTER DELETE ON '||TableName||
         ' REFERENCING OLD ROW AS odr'||
         ' FOR EACH ROW'||
         ' BEGIN ATOMIC';

      SET stmt = stmt||
         'UPDATE HS_TBL_'||TableName||
         ' SET HS_Hist.HS_HistoryEndTime = CURRENT_TIMESTAMP'||
         ' WHERE '||tmpAllPKeyComparison_odr||
         ' AND HS_Hist.HS_HistoryBeginTime ='||
         ' (SELECT MAX( HS_Hist.HS_HistoryBeginTime )'||
         ' FROM HS_TBL_'||TableName||
         ' WHERE '||tmpAllPKeyComparison_odr||
         ' AND HS_Hist.HS_HistoryBeginTime <'||
         ' CURRENT_TIMESTAMP);';

      SET stmt = stmt||' END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateDeleteTrigger(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateDeleteTrigger(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

### 6.1.8 HS_CreateHTForbidUpdateHistoryRowsTrigger Procedure

**Purpose**

Generate a trigger toforbid updating other than the latest history row in a certain history row set when value of the HS_Hist column of the history table is updating.

**Definition**

```
CREATE PROCEDURE HS_CreateHTForbidUpdateHistoryRowsTrigger(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllPKeyComparison_nwr CLOB;
      DECLARE tmpAllPKeyComparison_odr CLOB;

      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, NULL, NULL, tmpAllColumns);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, NULL, 'nwr.', tmpAllPKeyComparison_nwr);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, NULL, 'odr.', tmpAllPKeyComparison_odr);

      SET stmt = 'BEGIN'
      SET stmt = stmt||
         ' CREATE TRIGGER HS_TR_HS_'||TableName||'_UPD_T1'||
         ' BEFORE UPDATE OF HS_Hist ON HS_TBL_'||TableName||
         ' REFERENCING OLD ROW AS odr'||
         ' NEW ROW AS nwr'||
         ' FOR EACH ROW'||
         ' BEGIN ATOMIC'||
         ' DECLARE nrow INTEGER;'||
         ' SET nrow ='||
         ' (SELECT COUNT(*)'||
         ' FROM HS_TBL_'||TableName||
         ' WHERE '||tmpAllPKeyComparison_nwr||
         ' AND HS_Hist.HS_HistoryBeginTime ='||
         '  nwr.HS_Hist.HS_HistoryEndTime'||
         ' AND HS_Hist.HS_HistoryBeginTime <>'||
         '  odr.HS_Hist.HS_HistoryEndTime);'||
         ' IF nrow <> 1'||
         ' AND odr.HS_Hist.HS_HistoryBeginTime <>'||
         '  (SELECT MAX(HS_Hist.HS_HistoryBeginTime)'||
         '   FROM HS_TBL_'||TableName||
         '   WHERE '||tmpAllPKeyComparison_odr||') THEN'||
         ' SIGNAL XXXXX SET MESSAGE_TEXT ='||
         ' \'Target history row is not latest.\';'||
         ' END IF;'||
         ' END;';
      SET stmt = stmt||'END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateHTForbidUpdateHistoryRowsTrigger(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateHTForbidUpdateHistoryRowsTrigger(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

3) The following trigger isgenerated in the *HS_CreateHTForbidUpdateHistoryRowsTrigger(information_schema.SQL_IDENTIFIER)* procedure:

   a) a trigger which forbids updating other than a history row of the latest in a certain history row set when value of the HS_Hist column of the history table is updating. However, updating is permitted when it is applied to the below-mentioned case of 3)-a)-ii) in the next section.

#### 6.1.8+1 HS_CreateHTConsistencyMaintenanceTrigger Procedure

**Purpose**

Generate a trigger to preserve the consistency of history begin time and history end time stored in the history table.

**Definition**

```
CREATE PROCEDURE HS_CreateHTConsistencyMaintenanceTrigger(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllPKeyComparison_nwr CLOB;
      DECLARE tmpAllPKeyComparison_odr CLOB;

      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, NULL, NULL, tmpAllColumns);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, NULL, 'nwr.', tmpAllPKeyComparison_nwr);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, NULL, 'odr.', tmpAllPKeyComparison_odr);

      SET stmt = 'BEGIN'
      SET stmt = stmt||
         ' CREATE TRIGGER HS_TR_HS_'||TableName||'_UPD_T2'||
         ' AFTER UPDATE OF HS_Hist ON HS_TBL_'||TableName||
         ' REFERENCING OLD ROW AS odr'||
         ' NEW ROW AS nwr'||
         ' FOR EACH ROW'||
         ' BEGIN ATOMIC'||
         ' IF odr.HS_Hist.HS_HistoryEndTime IS NOT NULL'||
         ' AND odr.HS_Hist.HS_HistoryBeginTime <>'||
         '  nwr.HS_Hist.HS_HistoryBeginTime THEN'||
         ' SIGNAL XXXXX SET MESSAGE_TEXT ='||
         ' \'History end time of history row is already set.\';'||
         ' ELSEIF odr.HS_Hist.HS_HistoryEndTime IS NOT NULL'||
         ' AND nwr.HS_Hist.HS_HistoryEndTime IS NULL THEN'||
         ' SIGNAL XXXXX SET MESSAGE_TEXT ='||
         ' \'Given history end time must not be null.\';'||
         ' ELSEIF nwr.HS_Hist.HS_HistoryBeginTime >='||
         '  nwr.HS_Hist.HS_HistoryEndTime THEN'||
         ' SIGNAL XXXXX SET MESSAGE_TEXT ='||
         ' \'history end time precedes history begin time.\';'||
         ' END IF;'||
         ' IF EXISTS(SELECT *'||
         ' FROM HS_TBL_'||TableName||
         ' WHERE HS_Hist.HS_HistoryEndTime ='||
         '  odr.HS_Hist.HS_HistoryBeginTime'||
         ' AND HS_Hist.HS_HistoryEndTime <>'||
         '  nwr.HS_Hist.HS_HistoryBeginTime) THEN'||
         ' UPDATE HS_TBL_'||TableName||
         ' SET HS_Hist.HS_HistoryEndTime ='||
         '  nwr.HS_Hist.HS_HistoryBeginTime'||
         ' WHERE '||tmpAllPKeyComparison_odr||
         ' AND HS_Hist.HS_HistoryEndTime ='||
         '  odr.HS_Hist.HS_HistoryBeginTime'||
```

```
        ' AND HS_Hist.HS_HistoryEndTime <>'||
        '  nwr.HS_Hist.HS_HistoryBeginTime;'||
        ' END IF;'||
        ' END;';
    SET stmt = stmt||'END';

    EXECUTE IMMEDIATE stmt;
END
```

**Definitional Rules**

1) The *HS_CreateHTConsistencyMaintenanceTrigger(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateHTConsistencyMaintenanceTrigger (information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

3) The following trigger is generated in the *HS_CreateHTConsistencyMaintenanceTrigger (information_schema.SQL_IDENTIFIER)* procedure:

   a) a trigger which executes the following operation when value of the HS_Hist column of the history table is updated:

      i) If a value which the turn of time reverses was set, then an exception condition is raised.

      ii) When history begin time of the latest history row is updated, the value of history end time of history row in front of it is set up, and adjustment is maintained.

### 6.1.9 HS_CreateHistoryTimeMethod Procedure

**Purpose**

Generate methods to set the value of history begin time and history end time of the latest history row in the specified history row set.

**Definition**

```
CREATE PROCEDURE HS_CreateHistoryTimeMethod(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllPKeyColumnsDef_HS_PRM CLOB;
      DECLARE tmpAllPKeyComparison_HS_PRM CLOB;

      CALL HS_CreateCommaSeparatedPrimaryKeyAndTypeList(
         TableName, 'HS_PRM_', NULL, tmpAllPKeyColumnsDef_HS_PRM);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, 'HS_PRM_', NULL, tmpAllPKeyComparison_HS_PRM);

      SET stmt = 'BEGIN ATOMIC ';

      SET stmt = stmt||
         'CREATE STATIC METHOD HS_HistoryBeginTime('||
         tmpAllPKeyColumnsDef_HS_PRM||', HS_BeginOfPeriod TIMESTAMP)'||
         ' RETURNS TIMESTAMP'||
         ' FOR HS_TYPE_'||TableName||
         ' BEGIN'||
         '  UPDATE HS_TBL_'||TableName||
         '   SET HS_Hist.HS_HistoryBeginTime = HS_BeginOfPeriod'||
         '   WHERE '||tmpAllPKeyComparison_HS_PRM||
         '    AND HS_Hist.HS_HistoryBeginTime ='||
         '     (SELECT MAX(HS_Hist.HS_HistoryBeginTime)'||
         '      FROM HS_TBL_'||TableName||
         '      WHERE '||tmpAllPKeyComparison_HS_PRM||');'||
         '  RETURN HS_BeginOfPeriod;'||
         ' END;';
      SET stmt = stmt||
         'CREATE STATIC METHOD HS_HistoryEndTime('||
         tmpAllPKeyColumnsDef_HS_PRM||', HS_EndOfPeriod TIMESTAMP)'||
         ' RETURNS TIMESTAMP'||
         ' FOR HS_TYPE_'||TableName||
         ' BEGIN'||
         '  UPDATE HS_TBL_'||TableName||
         '   SET HS_Hist.HS_HistoryEndTime = HS_EndOfPeriod'||
         '   WHERE '||tmpAllPKeyComparison_HS_PRM||
         '    AND HS_Hist.HS_HistoryBeginTime ='||
         '     (SELECT MAX(HS_Hist.HS_HistoryBeginTime)'||
         '      FROM HS_TBL_'||TableName||
         '      WHERE '||tmpAllPKeyComparison_HS_PRM||');'||
         '  RETURN HS_EndOfPeriod;'||
         ' END;';

      SET stmt = stmt||'END';
      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateHistoryTimeMethod(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreateHistoryTimeMethod(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

### 6.1.10 HS_CreatePeriodMethod Procedure

**Purpose**

Generate methods to obtain restructured history table which stores change history of the specified columns.

**Definition**

```
CREATE PROCEDURE HS_CreatePeriodMethod(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllColumns_HT1 CLOB;
      DECLARE tmpAllColumnsDef CLOB;
      DECLARE tmpAllPKeyColumns CLOB;
      DECLARE tmpAllPKeyComparison_HT1_HT2 CLOB;

      DECLARE i INTEGER;

      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, NULL, NULL, tmpAllColumns);
      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, 'HT1.', NULL, tmpAllColumns_HT1);
      CALL HS_CreateCommaSeparatedAllColumnAndTypeList(
         TableName, NULL, NULL, tmpAllColumnsDef);
      CALL HS_CreateCommaSeparatedPrimaryKeyList(
         TableName, NULL, NULL, tmpAllPKeyColumns);
      CALL HS_CreatePrimaryKeySelfJoinCondition(
         TableName, 'HT1.', 'HT2.', tmpAllPKeyComparison_HT1_HT2);

      SET stmt = 'BEGIN ';

      SET stmt = stmt||
         'CREATE STATIC METHOD HS_Period('||
         '  TargetColumn information_schema.SQL_IDENTIFIER)'||
         ' RETURNS TABLE('||tmpAllColumnsDef||', HS_Hist HS_History)'||
         ' FOR HS_TYPE_'||TableName||
         ' BEGIN'||
         '  RETURN HS_Period(ARRAY[TargetColumn]);'||
         ' END;';
      SET stmt = stmt||
         'CREATE STATIC METHOD HS_Period('||
         '  TargetColumns information_schema.SQL_IDENTIFIER ARRAY)'||
         ' RETURNS TABLE('||tmpAllColumnsDef||', HS_Hist HS_History)'||
         ' FOR HS_TYPE_'||TableName||
         ' BEGIN';
      SET stmt = stmt||
         '  DECLARE tableName information_schema.SQL_IDENTIFIER;'||
         '  DECLARE periodColumnsComparison_HT1_HT2 CLOB;'||
         '  DECLARE tmpAllColumns CLOB;'||
         '  DECLARE tmpAllColumns_HT1 CLOB;'||
         '  DECLARE tmpPKeyComparison_HT1_HT2 CLOB;'||
         '  DECLARE stmt CLOB;'||
         '  DECLARE Result_Tbl ROW('||
         tmpAllColumnsDef||', HS_Hist HS_History) MULTISET;'||
         '  SET tableName = \''||TableName||'\';'||
```

```
   '  SET tmpAllColumns = \''||tmpAllColumns||'\';'||
   '  SET tmpAllColumns_HT1 = \''||tmpAllColumns_HT1||'\';'||
   '  SET tmpPKeyComparison_HT1_HT2 = \''||
   tmpAllPKeyComparison_HT1_HT2||'\';'||
   '  SET i = 1;'||
   '  WHILE i <= CARDINALITY(TargetColumns) DO'||
   '   IF i > 1 THEN'||
   '    SET periodColumnsComparison_HT1_HT2 ='||
   '     periodColumnsComparison_HT1_HT2||\' AND \';'||
   '   END IF;'||
   '   SET periodColumnsComparison_HT1_HT2 ='||
   '    periodColumnsComparison_HT1_HT2||\'(\''||'||
   '    \'HT1.\'||TargetColumns[i]||\' ='||
   '     HT2.\'||TargetColumns[i]||'||
   '    \' OR HT1.\'||TargetColumns[i]||\' IS NULL\''||'||
   '    \' AND HT2.\'||TargetColumns[i]||\' IS NULL)\';'||
   '  END WHILE;'||
   '  SET stmt ='||
   '   \'WITH RECURSIVE PeriodTemp(\''||'||
   '     tmpAllColumns||\', HS_Hist) AS\''||'||
   '   \' (\''||'||
   '   \'  SELECT \'||tmpAllColumns||\', HS_Hist\''||'||
   '   \'   FROM HS_TBL_\'||tableName||\'\''||'||
   '   \'  UNION ALL\''||'||
   '   \'  SELECT \'||tmpAllColumns_HT1||\',\''||'||
   '   \'         NEW HS_History(\''||'||
   '   \'           HT1.HS_Hist.HS_HistoryBeginTime,\''||'||
   '   \'           HT2.HS_Hist.HS_HistoryEndTime)\''||'||
   '   \'   FROM PeriodTemp HT1, HS_TBL_\'||tableName||\' HT2\''||'||
   '   \'   WHERE \'||tmpPKeyComparison_HT1_HT2||\'\''||'||
   '   \'    AND HT1.HS_Hist.HS_HistoryEndTime =\''||'||
   '   \'         HT2.HS_Hist.HS_HistoryBeginTime\''||'||
   '   \'    AND (\'||periodColumnsComparison_HT1_HT2||\')\''||'||
   '   \' )\''||'||
   '   \' SELECT \'||tmpAllColumns_HT1||\', HT1.HS_Hist\''||'||
   '   \'  FROM PeriodTemp HT1\''||'||
   '   \'  WHERE NOT EXISTS(\''||'||
   '   \'   SELECT *\''||'||
   '   \'    FROM PeriodTemp HT2\''||'||
   '   \'    WHERE \'||tmpPKeyComparison_HT1_HT2||\'\''||'||
   '   \'     AND (\'||periodColumnsComparison_HT1_HT2||\')\''||'||
   '   \'     AND (HT1.HS_Hist.HS_HistoryBeginTime >\''||'||
   '   \'           HT2.HS_Hist.HS_HistoryBeginTime\''||'||
   '   \'         AND (HT2.HS_Hist.HS_HistoryEndTime IS NULL\''||'||
   '   \'           OR HT1.HS_Hist.HS_HistoryBeginTime <\''||'||
   '   \'               HT2.HS_Hist.HS_HistoryEndTime)\''||'||
   '   \'       OR HT1.HS_Hist.HS_HistoryEndTime >\''||'||
   '   \'           HT2.HS_Hist.HS_HistoryBeginTime\''||'||
   '   \'         AND (HT2.HS_Hist.HS_HistoryEndTime IS NULL\''||'||
   '   \'           OR HT1.HS_Hist.HS_HistoryEndTime <\''||'||
   '   \'               HT2.HS_Hist.HS_HistoryEndTime)\''||'||
   '   \'      )\''||'||
   '   \'  )\';';
SET stmt = stmt||
   '  SET stmt = \'SET ? = TABLE(\'||stmt||\')\';'||
   '  PREPARE pstmt FROM stmt;'||
   '  EXECUTE pstmt INTO Result_Tbl;'||
   '  RETURN Result_Tbl;';
SET stmt = stmt||' END;';

SET stmt = stmt||'END';
```

```
      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreatePeriodMethod(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_CreatePeriodMethod(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

### 6.1.11 HS_InitializeHistoryTable Procedure

**Purpose**

Insert history rows corresponding to the rows stored in the current state table to the history table.

**Definition**

```
CREATE PROCEDURE HS_InitializeHistoryTable(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   NOT DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;

      CALL HS_CreateCommaSeparatedAllColumnList(
         TableName, NULL, NULL, tmpAllColumns);

      SET stmt =
         'INSERT INTO HS_TBL_'||TableName||
         '('||tmpAllColumns||',HS_Hist)'||
         ' SELECT '||tmpAllColumns||
         ', NEW HS_History(CURRENT_TIMESTAMP, NULL) FROM '||
         TableName;

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_InitializeHistoryTable(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure.

**Description**

1) The *HS_InitializeHistoryTable(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_CreateHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER ARRAY)* procedure is given to the input parameter *TableName*.

## 6.2 HS_DropHistory Procedure and Sub Procedures

### 6.2.1 HS_DropHistory Procedure

**Purpose**

The HS_DropHistory procedure calls some sub-procedures. These sub-procedures generate a backup of history table for the specified current state table, drop the methods to refer the history table, drop the triggers to maintain the history table, and so on.

**Definition**

```
CREATE PROCEDURE HS_DropHistory(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN BackupTableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      CALL HS_DropHistoryErrorCheck(TableName, BackupTableName);
      CALL HS_CreateBackupTable(TableName, BackupTableName);
      CALL HS_DropHistoryTableTypeMethod(TableName);
      CALL HS_DropHistoryTrigger(TableName);
      CALL HS_DropHistoryTable(TableName);
      CALL HS_DropHistoryTableType(TableName);
   END
```

**Description**

1) The *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) an information_schema.SQL_IDENTIFIER value *BackupTableName*.

2) The name of table to stop storing of history rows is given to the input parameter *TableName*.

3) The name of table to copy contents stored in the history table is given to the input parameter *BackupTableName*.

4) The *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure calls the following procedures in order:

   a) The *HS_DropHistoryErrorCheck* procedure,

   b) the *HS_CreateBackupTable* procedure,

   c) the *HS_DropHistoryTableTypeMethod* procedure,

   d) the *HS_DropHistoryTrigger* procedure,

   e) the *HS_DropHistoryTable* procedure,

   f) the *HS_DropHistoryTableType* procedure.

### 6.2.2 HS_DropHistoryErrorCheck Procedure

**Purpose**

Checks whether the input parameters of HS_DropHistory procedure arevalid.

**Definition**

```
CREATE PROCEDURE HS_DropHistoryErrorCheck(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN BackupTableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   READS SQL DATA
   BEGIN
      IF TableName IS NULL THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: TableName is null.';
      END IF;
      IF NOT EXISTS(SELECT *
         FROM information_schema.tables
         WHERE table_name = TableName) THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: Table '||TableName||' does not exist.';
      END IF;
      IF NOT EXISTS(SELECT *
         FROM information_schema.tables
         WHERE table_name = 'HS_TBL_'||TableName) THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: History table HS_TBL_'||
            TableName||' does not exist.';
      IF BackupTableName IS NULL THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: BackupTableName is null.';
      END IF;
      IF EXISTS(SELECT *
         FROM information_schema.tables
         WHERE table_name = BackupTableName) THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Illegal input: Table '||BackupTableName||' already exists.';
      END IF;
   END
```

**Definitional Rules**

1) The *HS_DropHistoryErrorCheck(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure.

**Description**

1) The *HS_DropHistoryErrorCheck(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) an information_schema.SQL_IDENTIFIER value *BackupTableName*.

2) The value passed as the first parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *TableName*.

3) The value passed as the second parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *BackupTableName*.

4) For the *HS_DropHistoryErrorCheck(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure:

   a) Case:

      i) If *TableName* is the null value, then an exception condition is raised: SQL/MM History exception – XXXXX.

      ii) If *TableName* is table name which does not exist in the database, then an exception condition is raised: SQL/MM History exception – XXXXX.

      iii) If *HS_TBL_<TableName>* is table name which does not exist in the database, then an exception condition is raised: SQL/MM History exception – XXXXX.

      iv) If *BackupTableName* is the null value, then an exception condition is raised: SQL/MM History exception – XXXXX.

      v) If *BackupTableName* is a table name which exists in the database, then an exception condition is raised: SQL/MM History exception – XXXXX.

      vi) Otherwise, the procedure is completed normally.

### 6.2.3 HS_CreateBackupTable Procedure

**Purpose**

Generate a backup table for the history table corresponding to the specified current state table.

**Definition**

```
CREATE PROCEDURE HS_CreateBackupTable(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN BackupTableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllColumnsDef CLOB;

      CALL HS_CreateCommaSeparatedAllColumnList(
         'HS_TBL_'||TableName, NULL, NULL, tmpAllColumns);
      CALL HS_CreateCommaSeparatedAllColumnAndTypeList(
         'HS_TBL_'||TableName, NULL, NULL, tmpAllColumnsDef);

      SET stmt = 'BEGIN ';

      SET stmt = stmt||'CREATE TABLE '||
         BackupTableName||'('||tmpAllColumnsDef||');';
      SET stmt = stmt||'INSERT INTO '||BackupTableName||
         '('||tmpAllColumns||') SELECT '||tmpAllColumns||
         ' FROM HS_TBL_'||TableName||';';

      SET stmt = stmt||'END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_CreateBackupTable(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure.

**Description**

1) The *HS_CreateBackupTable(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) an information_schema.SQL_IDENTIFIER value *BackupTableName*.

2) The value passed as the first parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *TableName*.

3) The value passed as the second parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *BackupTableName*.

### 6.2.4 HS_DropHistoryTableTypeMethod Procedure

**Purpose**

Drop methods provided in the *HS_TYPE_*<*TableName*> type.

**Definition**

```
CREATE PROCEDURE HS_DropHistoryTableTypeMethod(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      SET stmt = 'BEGIN ';

      SET stmt = stmt||
         'ALTER TYPE HS_TYPE_'||TableName||
         ' DROP STATIC METHOD HS_HistoryBeginTime'||
         ' FOR HS_TYPE_'||TableName||';'||
         'ALTER TYPE HS_TYPE_'||TableName||
         ' DROP STATIC METHOD HS_HistoryEndTime'||
         ' FOR HS_TYPE_'||TableName||';';
         'ALTER TYPE HS_TYPE_'||TableName||
         ' DROP STATIC METHOD HS_Period'||
         ' FOR HS_TYPE_'||TableName||';';

      SET stmt = stmt||'END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_DropHistoryTableTypeMethod(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure.

**Description**

1) The *HS_DropHistoryTableTypeMethod(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *TableName*.

### 6.2.5 HS_DropHistoryTrigger Procedure

**Purpose**

Drop triggers generated in the *HS_CreateHistory* procedure.

**Definition**

```
CREATE PROCEDURE HS_DropHistoryTrigger(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      SET stmt = 'BEGIN ';

      SET stmt = stmt||'DROP TRIGGER HS_TR_HS_'||TableName||'_UPD_T2;';
      SET stmt = stmt||'DROP TRIGGER HS_TR_HS_'||TableName||'_UPD_T1;';

      SET stmt = stmt||'DROP TRIGGER HS_TR_'||TableName||'_DEL;';
      SET stmt = stmt||'DROP TRIGGER HS_TR_'||TableName||'_INS;';
      SET stmt = stmt||'DROP TRIGGER HS_TR_'||TableName||'_UPD;';

      SET stmt = stmt||'END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_DropHistoryTrigger(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure.

**Description**

1) The *HS_DropHistoryTrigger(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *TableName*.

### 6.2.6 HS_DropHistoryTable Procedure

**Purpose**

Drop the history table corresponding to the specified current state table.

**Definition**

```
CREATE PROCEDURE HS_DropHistoryTable(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      SET stmt = 'BEGIN ';

      SET stmt = stmt||'DROP TABLE HS_TBL_'||TableName||';';

      SET stmt = stmt||'END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_DropHistoryTable(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure.

**Description**

1) The *HS_DropHistoryTable(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *TableName*.

### 6.2.7 HS_DropHistoryTableType Procedure

**Purpose**

Drop the *HS_TYPE_<TableName>* type generated in the *HS_CreateHistory* procedure..

**Definition**

```
CREATE PROCEDURE HS_DropHistoryTableType(
    IN TableName information_schema.SQL_IDENTIFIER
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE stmt CLOB;

      SET stmt = 'BEGIN ';

      SET stmt = stmt||'DROP TYPE HS_TYPE_'||TableName||';';

      SET stmt = stmt||'END';

      EXECUTE IMMEDIATE stmt;
   END
```

**Definitional Rules**

1) The *HS_DropHistoryTableType(information_schema.SQL_IDENTIFIER)* procedure should be called only in the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure.

**Description**

1) The *HS_DropHistoryTableType(information_schema.SQL_IDENTIFIER)* procedure takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TableName*.

2) The value passed as the first parameter of the *HS_DropHistory(information_schema.SQL_IDENTIFIER, information_schema.SQL_IDENTIFIER)* procedure is given to the input parameter *TableName*.

## 6.3 Utility Procedures for History

The procedures in this subsection execute process performed in sub-procedure of HS_CreateHistory procedure or HS_DropHistory procedure.

### 6.3.1 HS_CreateCommaSeparatedAllColumnList Procedure

**Purpose**

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

**Definition**

```
CREATE PROCEDURE HS_CreateCommaSeparatedAllColumnList(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN ColumnNamePrefix CLOB,
    IN ColumnNamePostfix CLOB,
    OUT ResultValue CLOB
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE AllColumns information_schema.SQL_IDENTIFIER ARRAY;

      DECLARE i INTEGER;

      SET AllColumns = ARRAY( SELECT column_name
         FROM information_schema.columns
         WHERE table_name = TableName
         ORDER BY ordinal_position );

      SET i = 1;
      SET ResultValue = '';
      WHILE i <= CARDINALITY(AllColumns) 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||AllColumns[i];
         IF ColumnNamePostfix IS NOT NULL THEN
            SET ResultValue = ResultValue||ColumnNamePostfix;
         END IF;
         SET i = i + 1;
      END WHILE;
   END
```

**Description**

1) The *HS_CreateCommaSeparatedAllColumnList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) a CLOB value *ColumnNamePrefix*,

   c) a CLOB value *ColumnNamePostfix*,

2) Target table name to generate comma-separated list of the text, which concatenate the specified prefix, name of column and the specified postfix, is given to the input parameter *TableName*.

3) Prefix text to each name of columns is given to the input parameter *ColumnNamePrefix*.

4) Postfix text to each name of columns is given to the input parameter *ColumnNamePostfix*.

5) The *HS_CreateCommaSeparatedAllColumnList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

6) The text which is comma-separated list of the text, which concatenate the specified prefix, name of column of the specified table and the specified postfix, is stored to the output parameter *ResultValue*.

### 6.3.2 HS_CreateCommaSeparatedPrimaryKeyList Procedure

**Purpose**

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

**Definition**

```
CREATE PROCEDURE HS_CreateCommaSeparatedPrimaryKeyList(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN ColumnNamePrefix CLOB,
    IN ColumnNamePostfix CLOB,
    OUT ResultValue CLOB
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE AllPKeyColumns information_schema.SQL_IDENTIFIER ARRAY;

      DECLARE i INTEGER;

      SET AllPKeyColumns = ARRAY( SELECT column_name
         FROM information_schema.key_column_usage
         WHERE table_name = TableName
         ORDER BY ordinal_position );

      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||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(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) a CLOB value *ColumnNamePrefix*,

   c) a CLOB value *ColumnNamePostfix*,

2) Target table name to generate comma-separated list of the text, which concatenate the specified prefix, name of primary key column and the specified postfix, is given to the input parameter *TableName*.

3) Prefix text to each name of primary key columns is given to the input parameter *ColumnNamePrefix*.

4) Postfix text to each name of primary key columns is given to the input parameter *ColumnNamePostfix*.

5) The *HS_CreateCommaSeparatedPrimaryKeyList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

6) The text which is comma-separated list of the text, which concatenate the specified prefix, name of primary key column of the specified table and the specified postfix, is stored to the output parameter *ResultValue*.

### 6.3.3 HS_CreateCommaSeparatedAllColumnAndTypeList Procedure

**Purpose**

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

**Definition**

```
CREATE PROCEDURE HS_CreateCommaSeparatedAllColumnAndTypeList(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN ColumnDefPrefix CLOB,
    IN ColumnDefPostfix CLOB,
    OUT ResultValue CLOB
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE AllColumnsAndTypes CLOB ARRAY;

      DECLARE i INTEGER;

      --
      -- !! See Description
      --       about "AllColumnsAndTypes"
      --

      SET i = 1;
      SET ResultValue = '';
      WHILE i <= CARDINALITY(AllColumnsAndTypes) DO
         IF i > 1 THEN
            SET ResultValue = ResultValue||', ';
         END IF
         IF ColumnDefPrefix IS NOT NULL THEN
            SET ResultValue = ResultValue||ColumnDefPrefix;
         END IF;
         SET ResultValue = ResultValue||AllColumnsAndTypes[i];
         IF ColumnDefPostfix IS NOT NULL THEN
            SET ResultValue = ResultValue||ColumnDefPostfix;
         END IF;
         SET i = i + 1;
      END WHILE;
   END
```

**Description**

1) The *HS_CreateCommaSeparatedAllColumnAndTypeList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) a CLOB value *ColumnDefPrefix*,

   c) a CLOB value *ColumnDefPostfix*,

2) Target table name to generate comma-separated list of the text, which concatenate the specified prefix, name of column, white space, name of data type or name of domain and the specified postfix, is given to the input parameter *TableName*.

3) Prefix text to each text which concatenate name of column, white space and name of data type or name of domain is given to the input parameter *ColumnNamePrefix*.

4) Postfix text to each text which concatenate name of column, white space and name of data type or name of domain is given to the input parameter *ColumnDefPostfix*.

5) For the array *AllColumnsAndTypes*, the text which concatenate name of column of the specified table, white space and the name of data type or the name of domain is stored to each element of the array.

6) The *HS_CreateCommaSeparatedAllColumnAndTypeList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

7) The text which is comma-separated list of the text, which concatenate the specified prefix, the name of column of the specified table, white space, name of data type or name of domain and the specified postfix, is stored to the output parameter *ResultValue*.

### 6.3.4 HS_CreateCommaSeparatedPrimaryKeyAndTypeList Procedure

**Purpose**

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

**Definition**

```
CREATE PROCEDURE HS_CreateCommaSeparatedPrimaryKeyAndTypeList(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN ColumnDefPrefix CLOB,
    IN ColumnDefPostfix CLOB,
    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
         IF ColumnDefPrefix IS NOT NULL THEN
            SET ResultValue = ResultValue||ColumnDefPrefix;
         END IF;
         SET ResultValue = ResultValue||AllPKeyColumnsAndTypes[i];
         IF ColumnDefPostfix IS NOT NULL THEN
            SET ResultValue = ResultValue||ColumnDefPostfix;
         END IF;
         SET i = i + 1;
      END WHILE;
   END
```

**Description**

1) The *HS_CreateCommaSeparatedPrimaryKeyAndTypeList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) a CLOB value *ColumnDefPrefix*,

   c) a CLOB value *ColumnDefPostfix*,

2) Target table name to generate comma-separated list of the text, which concatenate the specified prefix, name of primary key column, white space, name of data type or name of domain and the specified postfix, is given to the input parameter *TableName*.

3) Prefix text to each text which concatenate name of primary key column, white space and name of data type or name of domain is given to the input parameter *ColumnNamePrefix*.

4) Postfix text to each text which concatenate name of primary key column, white space and name of data type or name of domain is given to the input parameter *ColumnDefPostfix*.

5) For the array *AllPKeyColumnsAndTypes*, the text which concatenate name of primary key column of the specified table, white space and the name of data type or the name of domain is stored to each element of the array.

6) The *HS_CreateCommaSeparatedPrimaryKeyAndTypeList(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

7) The text which is comma-separated list of the text, which concatenate the specified prefix, the name of primary key column of the specified table, white space, name of data type or name of domain and the specified postfix, is stored to the output parameter *ResultValue*.

### 6.3.5 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 concatenate the specified prefix text, name of column, an equal sign, the another specified prefix text and name of column.

**Definition**

```
CREATE PROCEDURE HS_CreatePrimaryKeySelfJoinCondition(
    IN TableName information_schema.SQL_IDENTIFIER,
    IN ColumnNamePrefix1 CLOB,
    IN ColumnNamePrefix2 CLOB,
    OUT ResultValue CLOB
   )
   LANGUAGE SQL
   DETERMINISTIC
   MODIFIES SQL DATA
   BEGIN
      DECLARE AllPKeyColumns information_schema.SQL_IDENTIFIER ARRAY;

      DECLARE i INTEGER;

      SET AllPKeyColumns = ARRAY( SELECT column_name
         FROM information_schema.key_column_usage
         WHERE table_name = TableName
         ORDER BY ordinal_position );

      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||AllPKeyColumns[i];

         SET ResultValue = ResultValue||' = ';

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

         SET i = i + 1;
      END WHILE;
   END
```

**Description**

1) The *HS_CreatePrimaryKeySelfJoinCondition(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following input parameters:

   a) an information_schema.SQL_IDENTIFIER value *TableName*,

   b) a CLOB value *ColumnNamePrefix1*,

   c) a CLOB value *ColumnNamePrefix2*,

2) Target table name to generate a text of self-join condition is given to the input parameter *TableName*.

3) Prefix text to the column name in the left side of the equal predicate is given to the input parameter *ColumnNamePrefix1*.

4) Prefix text to the column name in the right side of the equal predicate is given to the input parameter *ColumnNamePrefix2*.

5) The *HS_CreatePrimaryKeySelfJoinCondition(information_schema.SQL_IDENTIFIER, CLOB, CLOB, CLOB)* procedure takes the following output parameter:

   a) a CLOB value *ResultValue*.

6) The text of self-join condition for the specified table is stored to the output parameter *ResultValue*. The result text is a list of equal predicate separated by ' AND'. Each equal predicate is a text which concatenate the specified prefix text, name of column, an equal sign, the another specified prefix text and name of column.

# 7 History Types

## 7.1 HS_History Type and Routines

### 7.1.1 HS_History Type

**Purpose**

The *HS_History* type provides the definition of attributes of history row.

**Definition**

```
CREATE TYPE HS_History AS(
   HS_HistoryBeginTime TIMESTAMP,
   HS_HistoryEndTime TIMESTAMP)

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

   METHOD HS_Overlaps(beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
      RETURNS BOOLEAN,

   METHOD HS_Overlaps(hs_hist HS_History)
      RETURNS BOOLEAN,

   METHOD HS_Meets(timePoint TIMESTAMP)
      RETURNS BOOLEAN,

   METHOD HS_Meets(hs_hist HS_History)
      RETURNS BOOLEAN,

   METHOD HS_Precedes(timePoint TIMESTAMP)
      RETURNS BOOLEAN,

   METHOD HS_Precedes(hs_hist HS_History)
      RETURNS BOOLEAN,

   METHOD HS_Contains(timePoint TIMESTAMP)
      RETURNS BOOLEAN,

   METHOD HS_Contains(beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
      RETURNS BOOLEAN,

   METHOD HS_Contains(hs_hist HS_History)
      RETURNS BOOLEAN,

   METHOD HS_Equals(beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
      RETURNS BOOLEAN,

   METHOD HS_Equals(hs_hist HS_History)
      RETURNS BOOLEAN,

   METHOD HS_MonthInterval()
      RETURNS INTERVAL YEAR TO MONTH

   METHOD HS_DayInterval()
      RETURNS INTERVAL DAY TO SECOND,
```

```
METHOD HS_Intersect(beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
   RETURNS HS_History,

METHOD HS_Intersect(hs_hist HS_History)
   RETURNS HS_History
```

**Description**

1) The *HS_History* type provides for public use:

   a) a method *HS_Overlaps(TIMESTAMP, TIMESTAMP)*,

   b) a method *HS_Overlaps(HS_History)*,

   c) a method *HS_Meets(TIMESTAMP)*,

   d) a method *HS_Meets(HS_History)*,

   e) a method *HS_Precedes(TIMESTAMP)*,

   f) a method *HS_Precedes(HS_History)*,

   g) a method *HS_Contains(TIMESTAMP)*,

   h) a method *HS_Contains(TIMESTAMP, TIMESTAMP)*,

   i) a method *HS_Contains(HS_History)*,

   j) a method *HS_Equals(TIMESTAMP, TIMESTAMP)*,

   k) a method *HS_Equals(HS_History)*,

   l) a method *HS_MonthInterval()*,

   m) a method HS_DayInterval(),

   n) a method *HS _Intersect(TIMESTAMP, TIMESTAMP)*,

   o) a method *HS _Intersect(HS_History)*.

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

   a) a TIMESTAMP value *HS_HistoryBeginTime*,

   b) a TIMESTAMP value *HS_HistoryEndTime*.

### 7.1.2 HS_History Method

**Purpose**

Generate HS_History value which has the specified TIMESTAMP values as the history begin time and the history end time.

**Definition**

```
CREATE CONSTRUCTOR METHOD HS_History(
    beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
   RETURNS HS_History
   SELF AS RESULT
   FOR HS_History
   BEGIN
      IF beginOfPeriod IS NULL THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Invalid value: beginOfPeriod is null.';
      ELSEIF beginOfPeriod >= endOfPeriod THEN
         SIGNAL XXXXX SET MESSAGE_TEXT =
            'Invalid value: endOfPeriod precedes beginOfPeriod.';
      END IF;
      SET HS_HistoryBeginTime = beginOfPeriod;
      SET HS_HistoryEndTime = endOfPeriod;
   END
```

**Description**

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

   a) a TIMESTAMP value *beginOfPeriod*,

   b) a TIMESTAMP value *endOfPeriod*.

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

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

4) For the *HS_History(TIMESTAMP, TIMESTAMP)* method:

   a) Case:

      i) If *beginOfPeriod* is the null value, then an exception condition is raised: SQL/MM History exception – XXXXX.

      ii) If the time stored in *endOfPeriod* precedes the time stored in *beginOfPeriod*, then an exception condition is raised: SQL/MM History exception – XXXXX.

      iii) Otherwise, generate HS_History value which has the specified TIMESTAMP values as the history begin time and the history end time.

### 7.1.3 HS_Overlaps Methods

**Purpose**

Tests whether the period which a certain HS_History value expresses overlaps with the period which two specified TIMESTAMP values or a HS_History value expresses.

**Definition**

```
CREATE METHOD HS_Overlaps
   (beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (beginOfPeriod, endOfPeriod) OVERLAPS (s_bt, s_et);
   END
CREATE METHOD HS_Overlaps
   (hs_hist HS_History)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      SET p_bt = hs_hist.HS_HistoryBeginTime;
      SET p_et = COALESCE(hs_hist.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (p_bt, p_et) OVERLAPS (s_bt, s_et);
   END
```

**Description**

1) The *HS_Overlaps(TIMESTAMP, TIMESTAMP)* method takes the following input parameters:

   a) a TIMESTAMP value *beginOfPeriod*,

   b) a TIMESTAMP value *endOfPeriod*.

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

   a) an *HS_History* value *hs_hist*.

### 7.1.4 HS_Meets Methods

**Purpose**

Tests whether the end time of the period which a certain HS_History value expresses is equal to the specified TIMESTAMP value or the begin time of the period which a HS_History value expresses.

**Definition**

```
CREATE METHOD HS_Meets
   (timePoint TIMESTAMP)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_et TIMESTAMP;

      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_et = timePoint);
   END
CREATE METHOD HS_Meets
   (hs_hist HS_History)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_et TIMESTAMP;

      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_et = hs_hist.HS_HistoryBeginTime);
   END
```

**Description**

1) The *HS_Meets(TIMESTAMP)* method takes the following input parameter:

   a) a TIMESTAMP value *timePoint*.

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

   a) an *HS_History* value *hs_hist*.

### 7.1.5 HS_Precedes Methods

**Purpose**

Tests whether the whole period which a certain HS_History value expresses precedes the specified TIMESTAMP value or the begin time of the period which a HS_History value expresses.

**Definition**

```
CREATE METHOD HS_Precedes
   (timePoint TIMESTAMP)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_et TIMESTAMP;

      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_et <= timePoint);
   END
CREATE METHOD HS_Precedes
   (hs_hist HS_History)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_et TIMESTAMP;

      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_et <= hs_hist.HS_HistoryBeginTime);
   END
```

**Description**

1) The *HS_Precedes(TIMESTAMP)* method takes the following input parameter:

   a) a TIMESTAMP value *timePoint*.

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

   a) an *HS_History* value *hs_hist*.

### 7.1.6 HS_Contains Methods

**Purpose**

Tests whether the period which a certain HS_History value expresses contains the specified TIMESTAMP value, the period which two specified TIMESTAMP value expresses or the whole period which a HS_History value expresses.

**Definition**

```
CREATE METHOD HS_Contains
   (timePoint TIMESTAMP)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (timePoint >= s_bt AND timePoint < s_et);
   END

CREATE METHOD HS_Contains
   (beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      IF beginOfPeriod > endOfPeriod THEN
         SET p_bt = endOfPeriod;
         SET p_et = beginOfPeriod;
      ELSE
         SET p_bt = beginOfPeriod;
         SET p_et = endOfPeriod;
      END IF;

      RETURN (p_bt >= s_bt AND p_et <= s_et AND p_bt < s_et);
   END

CREATE METHOD HS_Contains
   (hs_hist HS_History)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      SET p_bt = hs_hist.HS_HistoryBeginTime;
```

```
      SET p_et = COALESCE(hs_hist.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (p_bt >= s_bt AND p_et <= s_et);
   END
```

**Description**

1) The *HS_Contains(TIMESTAMP)* method takes the following input parameter:

   a) a TIMESTAMP value *timePoint*.

2) The *HS_Contains(TIMESTAMP, TIMESTAMP)* method takes the following input parameters:

   a) a TIMESTAMP value *beginOfPeriod*,

   b) a TIMESTAMP value *endOfPeriod*.

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

   a) an *HS_History* value *hs_hist*.

### 7.1.7 HS_Equals Methods

**Purpose**

Tests whether the period which a certain HS_History value expresses is equal to the period which two specified TIMESTAMP value or a HS_History value expresses.

**Definition**

```
CREATE METHOD HS_Equals
   (beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      IF beginOfPeriod > endOfPeriod THEN
         SET p_bt = endOfPeriod;
         SET p_et = beginOfPeriod;
      ELSE
         SET p_bt = beginOfPeriod;
         SET p_et = endOfPeriod;
      END IF;

      RETURN (s_bt = p_bt AND s_et = p_et);
   END
CREATE METHOD HS_Equals
   (hs_hist HS_History)
   RETURNS BOOLEAN
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      SET p_bt = hs_hist.HS_HistoryBeginTime;
      SET p_et = COALESCE(hs_hist.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_bt = p_bt AND s_et = p_et);
   END
```

**Description**

1) The *HS_Equals(TIMESTAMP, TIMESTAMP)* method takes the following input parameters:

   a) a TIMESTAMP value *beginOfPeriod*,

   b) a TIMESTAMP value *endOfPeriod*.

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

a) an *HS_History* value *hs_hist*:

### 7.1.8 HS_MonthInterval Method

**Purpose**

Obtain length of the period which a certain HS_History value expresses as year-month time interval.

**Definition**

```
CREATE METHOD HS_MonthInterval()
   RETURNS INTERVAL YEAR TO MONTH
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_et - s_bt) YEAR TO MONTH;
   END
```

**Description**

1) For the *HS_MonthInterval()* method:

   a) Case:

      i) If *HS_HistoryEndTime* is the null value, then difference between CURRENT_TIMESTAMP and value of *HS_HistoryBeginTime* attribute is returned.

      ii) Otherwise, difference between value of *HS_HistoryEndTime* attribute and value of *HS_HistoryBeginTime* attribute is returned.

### 7.1.9 HS_DayInterval Method

**Purpose**

Obtain length of the period which a certain HS_History value expresses as day-time time interval.

**Definition**

```
CREATE METHOD HS_DayInterval()
   RETURNS INTERVAL DAY TO SECOND
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      RETURN (s_et - s_bt) DAY TO SECOND;
   END
```

**Description**

1) For the *HS_DayInterval()* method:

   a) Case:

      i) If *HS_HistoryEndTime* is the null value, then difference between CURRENT_TIMESTAMP and value of *HS_HistoryBeginTime* attribute is returned.

      ii) Otherwise, difference between value of *HS_HistoryEndTime* attribute and value of *HS_HistoryBeginTime* attribute is returned.

### 7.1.10 HS_Intersect Methods

**Purpose**

Generate a new HS_History value with period which is overlap of the period which a certain HS_History value expresses and the period which two specified TIMESTAMP value or the specified HS_History value expresses.

**Definition**

```
CREATE METHOD HS_Intersect(
    beginOfPeriod TIMESTAMP, endOfPeriod TIMESTAMP)
   RETURNS HS_History
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      DECLARE st TIMESTAMP;
      DECLARE et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      IF beginOfPeriod > endOfPeriod THEN
         SET p_bt = endOfPeriod;
         SET p_et = beginOfPeriod;
      ELSE
         SET p_bt = beginOfPeriod;
         SET p_et = endOfPeriod;
      END IF;

      SET st = CASE
                  WHEN s_bt > p_bt THEN s_bt
                  ELSE p_bt
               END;
      SET et = CASE
                  WHEN s_et < p_et THEN s_et
                  ELSE p_et
               END;
      IF st < et THEN
         RETURN NEW HS_History(st, et);
      ELSE
         RETURN (NULL AS HS_History);
      END IF;
   END

CREATE METHOD HS_Intersect(hs_hist HS_History)
   RETURNS HS_History
   FOR HS_History
   BEGIN
      DECLARE s_bt TIMESTAMP;
      DECLARE s_et TIMESTAMP;
      DECLARE p_bt TIMESTAMP;
      DECLARE p_et TIMESTAMP;

      DECLARE st TIMESTAMP;
      DECLARE et TIMESTAMP;

      SET s_bt = SELF.HS_HistoryBeginTime;
      SET s_et = COALESCE(SELF.HS_HistoryEndTime, CURRENT_TIMESTAMP);
```

```
      SET p_bt = hs_hist.HS_HistoryBeginTime;
      SET p_et = COALESCE(hs_hist.HS_HistoryEndTime, CURRENT_TIMESTAMP);

      SET st = CASE
                  WHEN s_bt > p_bt THEN s_bt
                  ELSE p_bt
               END;
      SET et = CASE
                  WHEN s_et < p_et THEN s_et
                  ELSE p_et
               END;
      IF st < et THEN
         RETURN NEW HS_History(st, et);
      ELSE
         RETURN (NULL AS HS_History);
      END IF;
   END
```

**Description**

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

   a) a TIMESTAMP value *beginOfPeriod*,

   b) a TIMESTAMP value *endOfPeriod*.

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

   a) an *HS_History* values *hs_hist*.

## 7.2 HS_TYPE_<TableName> Type and Routines

### 7.2.1 HS_TYPE_<TableName> Type

**Purpose**

The *HS_History* type provides the definition of history row corresponding to the specified current state table.

**Definition**

```
CREATE TYPE HS_TYPE_<TableName> AS(
   <AllColumnsDef>, HS_Hist HS_History)

   STATIC METHOD HS_Period(
    TargetColumns information_schema.SQL_IDENTIFIER)
      RETURNS TABLE(<AllColumnsDef>, HS_Hist HS_History),

   STATIC METHOD HS_Period(
    TargetColumns information_schema.SQL_IDENTIFIER ARRAY)
      RETURNS TABLE(<AllColumnsDef>, HS_Hist HS_History),

   STATIC METHOD HS_HistoryBeginTime(
    <AllPKeyColumnsDef>, HS_BeginOfPeriod TIMESTAMP)
      RETURNS TIMESTAMP,

   STATIC METHOD HS_HistoryEndTime(
    <AllPKeyColumnsDef>, HS_EndOfPeriod TIMESTAMP)
      RETURNS TIMESTAMP
```

**Description**

1) The *HS_TYPE_<TableName>* type provides for public use:

   a) a method *HS_Period(information_schema.SQL_IDENTIFIER ARRAY)*,

   b) a method *HS_HistoryBeginTime(<AllPKeyColumnsDef>, TIMESTAMP)*,

   c) a method *HS_HistoryEndTime(<AllPKeyColumnsDef>, TIMESTAMP)*.

2) *HS_TYPE_<TableName>* type has the following attributes:

   a) attributes correspond to columns of current state table,

   b) a *HS_History* value *HS_Hist*,

### 7.2.2 HS_HistoryBeginTime Method

**Purpose**

Set the value of history begin time of the latest history row in the specified history row set.

**Definition**

```
CREATE STATIC METHOD HS_HistoryBeginTime(
   <AllPKeyColumnsDef>, HS_BeginOfPeriod TIMESTAMP)
   RETURNS TIMESTAMP
   FOR HS_TYPE_<TableName>
   BEGIN
      UPDATE HS_TBL_<TableName>
         SET HS_Hist.HS_HistoryBeginTime = HS_BeginOfPeriod
         WHERE <AllPKeyComparison>
            AND HS_Hist.HS_HistoryBeginTime =
               (SELECT MAX(HS_Hist.HS_HistoryBeginTime)
                FROM HS_TBL_<TableName>
                WHERE <AllPKeyComparison>);
      RETURN HS_BeginOfPeriod;
   END
```

**Description**

1) The *HS_HistoryBeginTime(<AllPKeyColumnsDef, TIMESTAMP)* method takes the following input parameters:

   a) parameters correspond to primary key columns of current state table,

   b) a TIMESTAMP value *HS_BeginOfPeriod*.

2) The *HS_HistoryBeginTime(<AllPKeyColumnsDef, TIMESTAMP)* method returns a TIMESTAMP value which is given as the input parameter *HS_BeginOfPeriod*.

### 7.2.3 HS_HistoryEndTime Method

**Purpose**

Set the value of history end time of the latest history row in the specified history row set.

**Definition**

```
CREATE STATIC METHOD HS_HistoryEndTime(
   <AllPKeyColumnsDef>, HS_EndOfPeriod TIMESTAMP)
   RETURNS TIMESTAMP
   FOR HS_TYPE_<TableName>
   BEGIN
      UPDATE HS_TBL_<TableName>
         SET HS_Hist.HS_HistoryEndTime = HS_EndOfPeriod
         WHERE <AllPKeyComparison>
            AND HS_Hist.HS_HistoryBeginTime =
               (SELECT MAX(HS_Hist.HS_HistoryBeginTime)
                FROM HS_TBL_<TableName>
                WHERE <AllPKeyComparison>);
      RETURN HS_EndOfPeriod;
   END
```

**Description**

1) The *HS_HistoryEndTime(<AllPKeyColumnsDef, TIMESTAMP)* method takes the following input parameters:

   a) parameters correspond to primary key columns of current state table,

   b) a TIMESTAMP value *HS_EndOfPeriod*.

2) The *HS_HistoryEndTime(<AllPKeyColumnsDef, TIMESTAMP)* method returns a TIMESTAMP value which is given as the input parameter *HS_EndOfPeriod*.

### 7.2.4 HS_Period Method

**Purpose**

Obtain restructured history table which stores change history of the specified columns.

**Definition**

```
CREATE STATIC METHOD HS_Period(
    TargetColumn information_schema.SQL_IDENTIFIER)
   RETURNS TABLE(<AllColumnsDef>, HS_Hist HS_History)
   FOR HS_TYPE_<TableName>
   BEGIN
      RETURN HS_Period(ARRAY[TargetColumn]);
   END


CREATE STATIC METHOD HS_Period(
    TargetColumns information_schema.SQL_IDENTIFIER ARRAY)
   RETURNS TABLE(<AllColumnsDef>, HS_Hist HS_History)
   FOR HS_TYPE_<TableName>
   BEGIN
      DECLARE tableName information_schema.SQL_IDENTIFIER;
      DECLARE periodColumnsComparison_HT1_HT2 CLOB;
      DECLARE tmpAllColumns CLOB;
      DECLARE tmpAllColumns_HT1 CLOB;
      DECLARE tmpPKeyComparison_HT1_HT2 CLOB;
      DECLARE stmt CLOB;
      DECLARE Result_Tbl ROW(<AllColumnsDef>, HS_Hist HS_History) MULTISET;

      SET tableName = '<TableName>';
      SET tmpAllColumns = '<AllColumns>';
      SET tmpAllColumns_HT1 = '<AllColumns_HT1>';
      SET tmpPKeyComparison_HT1_HT2 = '<AllPKeyComparison_HT1_HT2>';

      SET i = 1;
      WHILE i <= CARDINALITY(TargetColumns) DO
         IF i > 1 THEN
            SET periodColumnsComparison_HT1_HT2 =
               periodColumnsComparison_HT1_HT2||' AND ';
         END IF;
         SET periodColumnsComparison_HT1_HT2 =
            periodColumnsComparison_HT1_HT2||'('||
            'HT1.'||TargetColumns[i]||' = HT2.'||TargetColumns[i]||
            ' OR HT1.'||TargetColumns[i]||' IS NULL'||
            ' AND HT2.'||TargetColumns[i]||' IS NULL)';
      END WHILE;
      SET stmt =
         'WITH RECURSIVE PeriodTemp('||tmpAllColumns||', HS_Hist) AS'||
         ' ('||
         '  SELECT '||tmpAllColumns||', HS_Hist'||
         '   FROM HS_TBL_'||tableName||''||
         '  UNION ALL'||
         '  SELECT '||tmpAllColumns_HT1||','||
         '         NEW HS_History(HT1.HS_Hist.HS_HistoryBeginTime,'||
         '                        HT2.HS_Hist.HS_HistoryEndTime)'||
         '   FROM PeriodTemp HT1, HS_TBL_'||tableName||' HT2'||
         '   WHERE '||tmpPKeyComparison_HT1_HT2||''||
         '    AND HT1.HS_Hist.HS_HistoryEndTime ='||
         '         HT2.HS_Hist.HS_HistoryBeginTime'||
         '    AND ('||periodColumnsComparison_HT1_HT2||')'||
         ' )'||
```

```
        ' SELECT '||tmpAllColumns_HT1||', HT1.HS_Hist'||
        '  FROM PeriodTemp HT1'||
        '  WHERE NOT EXISTS('||
        '   SELECT *'||
        '    FROM PeriodTemp HT2'||
        '    WHERE '||tmpPKeyComparison_HT1_HT2||''||
        '     AND ('||periodColumnsComparison_HT1_HT2||')'||
        '     AND (HT1.HS_Hist.HS_HistoryBeginTime >'||
        '           HT2.HS_Hist.HS_HistoryBeginTime'||
        '         AND (HT2.HS_Hist.HS_HistoryEndTime IS NULL'||
        '           OR HT1.HS_Hist.HS_HistoryBeginTime <'||
        '               HT2.HS_Hist.HS_HistoryEndTime)'||
        '       OR HT1.HS_Hist.HS_HistoryEndTime >'||
        '           HT2.HS_Hist.HS_HistoryBeginTime'||
        '         AND (HT2.HS_Hist.HS_HistoryEndTime IS NULL'||
        '           OR HT1.HS_Hist.HS_HistoryEndTime <'||
        '               HT2.HS_Hist.HS_HistoryEndTime)'||
        '      )'||
        '  )';

    SET stmt = 'SET ? = TABLE('||stmt||')';
    PREPARE pstmt FROM stmt;
    EXECUTE pstmt INTO Result_Tbl;
    RETURN Result_Tbl;
END
```

**Description**

1) The *HS_Period(information_schema.SQL_IDENTIFIER)* method takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER value *TargetColumn*

2) The *HS_Period(information_schema.SQL_IDENTIFIER ARRAY)* method takes the following input parameter:

   a) an information_schema.SQL_IDENTIFIER ARRAY value *TargetColumns*.

3) The *HS_Period(information_schema.SQL_IDENTIFIER)* method returns the result which is returned as the result of the following method invocation:

   HS_Period(ARRAY[TargetColumn])

4) The table returned by the *HS_Period(information_schema.SQL_IDENTIFIER ARRAY)* method has the same schema as the history table. The table returned by the *HS_Period* method is a table which stores change history of only the specified columns. For the values of the columns which is not specified by the input parameter TargetColumns, the value in the time of either of the values of the column specified by the input parameter TargetColumns changing is stored.

# 8 SQL/MM History Information Schema

## 8.1 Introduction

# 9 SQL/MM History Definition Schema

## 9.1 Introduction

## 10 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 2 – 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 9075-2).

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).

**Table 2 – SQLSTATE class and subclass values**

| Category | Condition | Class | Subcondition | Subclass |
|---|---|---|---|---|
| X | SQL/MM History exception | 2F | incorrect input parameters | F01 |
| X | ??? | 2F | ??? | ??? |