

INTERNATIONAL
STANDARD

ISO/IEC
19075-2

First edition
2021-08

**Information technology — Guidance
for the use of database language
SQL —**

**Part 2:
Time-related information**

*Technologies de l'information — Recommandations pour l'utilisation
du langage de base de données SQL —*

Partie 2: Informations d'horodatage

IECNORM.COM : Click to view the full PDF of ISO/IEC 19075-2:2021



Reference number
ISO/IEC 19075-2:2021(E)

© ISO/IEC 2021

IECNORM.COM : Click to view the full PDF of ISO/IEC 19075-2:2021



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

	Page
Foreword.....	vi
Introduction.....	viii
1 Scope.....	1
2 Normative references.....	2
3 Terms and definitions.....	3
4 Time-related data types, constructs, operators, and predicates.....	4
4.1 Context of time-related specifications.....	4
4.2 Datetime types.....	4
4.3 DateTime literals.....	6
4.4 Interval types.....	7
4.5 Interval literals.....	8
4.6 Periods.....	9
4.7 Operations involving datetimes and intervals.....	10
4.8 Time-related predicates.....	10
4.8.1 Overlaps predicate.....	10
4.8.2 Period predicates.....	11
5 Time-related tables.....	14
5.1 Introduction to time-related tables.....	14
5.2 Application-time period tables.....	14
5.2.1 Introduction to application-time period tables.....	14
5.2.2 Extensions to primary key / unique constraints.....	14
5.2.3 Extensions to referential constraints.....	15
5.2.4 Inserting rows into tables containing an application-time period definition.....	18
5.2.5 Updating rows of tables containing an application-time period definition.....	18
5.2.6 Updating the table between specific points in time.....	19
5.2.7 Deleting rows from tables containing an application-time period definition.....	22
5.2.8 Deleting rows between specific points in time.....	22
5.2.9 Querying tables containing a period definition.....	24
5.2.10 Adding a period definition to a table.....	25
5.3 System-versioned tables.....	26
5.3.1 Introduction to system-versioned tables.....	26
5.3.2 Primary key and referential constraints.....	27
5.3.3 Updating rows in system-versioned tables.....	28
5.3.4 Deleting rows in system-versioned tables.....	29
5.3.5 Querying system-versioned tables.....	29
5.4 Bi-temporal tables.....	30
Bibliography.....	32

Index.....	33
------------	----

IECNORM.COM : Click to view the full PDF of ISO/IEC 19075-2:2021

Tables

Table	Page
1 Fields in datetime values	4
2 Mapping of datetime fields to datetime data types	5
3 Examples of the datetime data types	6
4 Examples of datetime literals	6
5 Fields in year-month INTERVAL values	7
6 Fields in day-time INTERVAL values	7
7 Examples of day-time INTERVAL values	8
8 Examples of interval literals	8
9 Valid operators involving datetimes and intervals	10
10 Example data table emp for primary key with application-time period	15
11 Example data table dept for foreign key with application-time period	16
12 Example data table emp for foreign key with application-time period	16
13 Content of table emp after insert with application-time period	18
14 Content of table emp before updating a row	18
15 Content of table emp after updating a row	19
16 Content of table emp before updating the application-time period of a row	19
17 Content of table emp after updating the application-time period of a row	19
18 Content of table emp before updating a row for an inner part of a period	20
19 Content of table emp after updating a row for an inner part of a period	20
20 Content of table emp before updating a row for all of a period	20
21 Content of table emp after updating a row for all of a period	21
22 Content of table emp before updating the edge of a period of a row	21
23 Content of table emp after updating the edge of a period of a row	21
24 Content of table emp before deleting an inner part of a period	22
25 Content of table emp with application-time period after deleting an inner part of a period	23
26 Content of table emp with application-time period before deleting a row	23
27 Content of table emp before deleting an edge of a period	23
28 Content of table emp after deleting an edge of a period	24
29 Content of system-versioned table emp before updating a row	28
30 Content of system-versioned table emp after updating a row	28
31 Content of system-versioned table emp before deleting a row	29
32 Content of system-versioned table emp after deleting a row	29

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents), or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This first edition of ISO/IEC 19075-2 cancels and replaces ISO/IEC TR 19075-2:2015.

This document is intended to be used in conjunction with the following editions of the parts of the ISO/IEC 9075 series:

- ISO/IEC 9075-1, sixth edition or later;
- ISO/IEC 9075-2, sixth edition or later;
- ISO/IEC 9075-3, sixth edition or later;
- ISO/IEC 9075-4, seventh edition or later;
- ISO/IEC 9075-9, fifth edition or later;
- ISO/IEC 9075-10, fifth edition or later;
- ISO/IEC 9075-11, fifth edition or later;
- ISO/IEC 9075-13, fifth edition or later;
- ISO/IEC 9075-14, sixth edition or later;
- ISO/IEC 9075-15, second edition or later;
- ISO/IEC 9075-16, first edition or later.

A list of all parts in the ISO/IEC 19075 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/-national-committees.

IECNORM.COM : Click to view the full PDF of ISO/IEC 19075-2:2021

Introduction

The organization of this document is as follows:

- 1) **Clause 1, "Scope"**, specifies the scope of this document.
- 2) **Clause 2, "Normative references"**, identifies additional standards that, through reference in this document, constitute provisions of this document.
- 3) **Clause 3, "Terms and definitions"**, defines the terms and definitions used in this document.
- 4) **Clause 4, "Time-related data types, constructs, operators, and predicates"**, explains time-related data types, operators, and predicates in SQL.
- 5) **Clause 5, "Time-related tables"**, explains how time-related tables are used.

IECNORM.COM : Click to view the full PDF of ISO/IEC 19075-2:2021

Information technology — Guidance for the use of database language SQL —

Part 2: Time-related information

1 Scope

This document describes the support in SQL for time-related information.

This document discusses the following features of the SQL language:

- Time-related data types
- Operations on time-related data
- Time-related Predicates
- Application-time period tables
- System-versioned tables
- Bi-temporal tables

IECNORM.COM : Click to view the full PDF of ISO/IEC 19075-2:2021

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

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

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9075-1 apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

4 Time-related data types, constructs, operators, and predicates

4.1 Context of time-related specifications

The requirements for the material discussed in this document shall be as specified in ISO/IEC 9075-1 and ISO/IEC 9075-2.

4.2 Datetime types

ISO/IEC 9075-2 defines the time-related data types, constructs, operators, and predicates described in this document. ISO/IEC 9075-2 specifies requirements for the material discussed in this document.

There are three *datetime types*, each of which is made up of different datetime fields.

A value of data type TIMESTAMP is made up of the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It is always a valid time at a valid Gregorian date.

A value of data type TIME comprises values of the datetime fields HOUR, MINUTE and SECOND. It is always a valid time of day.

A value of data type DATE is made up of the datetime fields YEAR, MONTH, and DAY. It is always a valid Gregorian date.

TIMESTAMP and TIME may be specified with a number of (decimal) digits of fractional seconds precision.

TIMESTAMP and TIME may also be specified as being WITH TIME ZONE, in which case every value has associated with it a time zone displacement (the offset of the TIMESTAMP or TIME from Coördinated Universal Time, UTC). In comparing values of a data type WITH TIME ZONE, the value of the time zone displacement is disregarded.

Table 1, “Fields in datetime values”, specifies the fields that can make up a datetime value.

Table 1 — Fields in datetime values

Keyword	Meaning
YEAR	Year, between 0001 and 9999
MONTH	Month within year, between 01 and 12
DAY	Day within month, between 1 and 31, but further constrained by the value of MONTH and YEAR fields, according to the rules for well-formed dates in the Gregorian calendar.
HOUR	Hour within day, between 00 and 23
MINUTE	Minute within hour, between 00 and 59

Keyword	Meaning
SECOND	Second and possibly fraction of a second within minute, between 00 and 61.999...
TIMEZONE_HOUR	Hour value of time zone displacement, between -14 and 14. The range for time zone intervals is larger than many readers might expect because it is governed by political decisions in governmental bodies rather than by any natural law.
TIMEZONE_MINUTE	Minute value of time zone displacement, between -59 and 59. When the value of TIMEZONE_HOUR is either -12 or 14, the value of TIMEZONE_MINUTE is restricted to be 00 (zeros).

There is an ordering of the significance of these fields. This is, from most significant to least significant: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

Table 2, “Mapping of datetime fields to datetime data types”, clarifies which of the various datetime fields can appear in each of the datetime data types.

Table 2 — Mapping of datetime fields to datetime data types

Data type	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND	TZ HOUR	TZ MINUTE
TIMESTAMP	Y	Y	Y	Y	Y	Y	N	N
TIMESTAMP WITH TIME ZONE	Y	Y	Y	Y	Y	Y	Y	Y
TIME	N	N	N	Y	Y	Y	N	N
TIME WITH TIME ZONE	N	N	N	Y	Y	Y	Y	Y
DATE	Y	Y	Y	N	N	N	N	N

The surface of the earth is divided into zones, called time zones, in which every correct clock tells the same time, known as *local time*. Local time is equal to UTC (Coordinated Universal Time) plus the *time zone displacement*, which is an interval value that ranges between INTERVAL '-12:00' HOUR TO MINUTE and INTERVAL '+14:00' HOUR TO MINUTE. The time zone displacement is constant throughout a time zone, changing at the beginning and end of Summer Time, where applicable.

NOTE 1 — [IANA Time Zone Database](#) is one resource from which SQL-implementations can determine details about time zones.

A datetime value, of data type TIME WITHOUT TIME ZONE or TIMESTAMP WITHOUT TIME ZONE, may represent a local time, whereas a datetime value of data type TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE represents UTC.

Table 3, “Examples of the datetime data types”, provides a few examples of datetime data type declarations.

Table 3 — Examples of the datetime data types

Data type	Explanation
TIMESTAMP (2)	This is a timestamp with a fractional precision of 2 for the seconds field
TIMESTAMP	This is a timestamp with no fractional precision for the seconds field
TIME (2)	This is a time with a fractional precision of 2 for the seconds field
TIME	This is a time with no fractional precision for the seconds field
DATE	This is a date

On occasion, UTC is adjusted by the omission of a second or the insertion of a “leap second” in order to maintain synchronization with sidereal time. This implies that sometimes, but very rarely, a particular minute will contain exactly 59, 61, or 62 seconds. Interval arithmetic that involves leap seconds or discontinuities in calendars will produce implementation-defined results.

For the convenience of users, whenever a datetime value with time zone is to be implicitly derived from one without (for example, in a simple assignment operation), SQL assumes the value without time zone to be local, subtracts the current default time zone displacement of the SQL-session from it to give UTC, and associates that time zone displacement with the result.

Conversely, whenever a datetime value without time zone is to be implicitly derived from one with, SQL assumes the value with time zone to be UTC, adds the time zone displacement to it to give local time, and the result, without any time zone displacement, is local.

Datetime data types will allow dates in the Gregorian format to be stored in the date range 0001-01-01 CE through 9999-12-31 CE.

NOTE 2 — Support for dates prior to 0001-01-01 CE is entirely implementation-defined. ISO/IEC 9075-2 neither specifies syntax or semantics for such dates, nor prohibits SQL-implementations from providing support. Additionally no support is provided for dates using other calendar systems (for example, Japanese imperial calendars) or *epochal* dating systems (e.g., geological eras).

4.3 Date Time literals

A datetime literal can specify datetime values of the respective datetime data types. A datetime literal consists of three parts. The keyword for the data type, the value in a fixed format and the timezone displacement. The format for the datetime literal is yyyy-mm-dd hh:mi:ss.ssss. The data type is automatically assigned to the literals depending on their content and the keyword used.

Table 4, “Examples of datetime literals”, gives examples of a few datetime literals.

Table 4 — Examples of datetime literals

Literal	Data type	Explanation
TIMESTAMP '2014-06-11 09:15:22.03'	TIMESTAMP (2)	This is a timestamp for the 11th of June 2014 at 9 hours, 15 minutes and 22.03 seconds
TIME'12:00:01+01:00'	TIME(0) WITH TIME ZONE	One second after noon in the timezone with a displacement of + 1 hour

Literal	Data type	Explanation
DATE '0001-01-01'	DATE	The first of January in year 1. This is the first possible date in SQL

4.4 Interval types

A value of an *interval type* represents the duration of a period of time. There are two classes of intervals. One class, called *year-month intervals*, has an interval precision that includes a YEAR field or a MONTH field, or both. The other class, called *day-time intervals*, has an express or implied interval precision that can include any set of contiguous fields other than YEAR or MONTH.

Table 5, “Fields in year-month INTERVAL values”, specifies the fields that make up a year-month interval.

Table 5 — Fields in year-month INTERVAL values

Keyword	Meaning	Valid values of INTERVAL fields
YEAR	Years	Unconstrained except by the leading field precision
MONTH	Months	Months (within years) (0-11)

Table 6, “Fields in day-time INTERVAL values”, specifies the fields that make up a day-time interval. A day-time interval is made up of a contiguous subset of those fields.

Table 6 — Fields in day-time INTERVAL values

Keyword	Meaning	Valid values of INTERVAL fields
DAY	Days	Unconstrained except by the leading field precision
HOUR	Hours	Hours (within days) (0-23)
MINUTE	Minutes	Minutes (within hours) (0-59)
SECOND	Seconds and possibly fractions of a second	Seconds (within minutes) (0-59.999...)

The actual subset of fields that comprise a value of either type of interval is called the precision of the value.

Within a value of type interval, the first field is constrained only by the precision of the leading field.

Values in interval fields other than SECOND are integers. SECOND, can be defined to have a precision of fractional seconds that indicates the number of decimal digits maintained following the decimal point in the seconds value.

Table 7, “Examples of day-time INTERVAL values”, gives examples of a few day-time interval literals.

Table 7 — Examples of day-time INTERVAL values

Data type	Explanation
INTERVAL YEAR TO MONTH	This is a year-month interval which is made up of the fields year and month
INTERVAL HOUR TO SECOND (2)	This is a day-time interval, which is made up of the fields hour, minute and second with a fractional precision of 2. The day value is always 0 (zero)
INTERVAL DAY	This is a day-time interval, which is made up of just the field day. All other fields are 0 (zero)
INTERVAL MONTH	This is a year-month interval which is made up of just the field month. The year has always the value 0 (zero)
INTERVAL SECOND (4)	This is a day-time interval, which is made up of just the field second with a fractional precision of 4. All other fields are 0 (zero)

Year-month intervals are comparable only with other year-month intervals. If two year-month intervals have different interval precisions, they are, for the purpose of any operations between them, converted to the same precision by appending new datetime fields to either one of the ends of one interval, or to both ends. New datetime fields are assigned a value of 0 (zero).

Day-time intervals are comparable only with other day-time intervals. If two day-time intervals have different interval precisions, they are, for the purpose of any operations between them, converted to the same precision by appending new datetime field to either one of the ends of one interval, or to both ends. New datetime fields are assigned a value of 0 (zero).

4.5 Interval literals

An interval literal can specify interval values of the respective interval data types. An interval literal consists of three parts. The keyword INTERVAL, the datetime interval in a fixed format and the interval qualifier. The format for the datetime literal is yyyy-mm-dd hh:mi:ss.ssss. An interval literal can be positive or negative. The data type is automatically assigned to the literal depending on the keywords used for the interval qualifier.

Table 8, “Examples of interval literals”, gives examples of a few datetime literals.

Table 8 — Examples of interval literals

Literal	Data type	Explanation
INTERVAL '1' MONTH	INTERVAL MONTH	One month
INTERVAL '01 10' DAY TO HOUR	INTERVAL DAY TO HOUR	One day and ten hours
INTERVAL '10:10:10.1' HOUR TO SECOND(1)	INTERVAL HOUR TO SECOND (1)	Ten hours, ten minutes and 10.1 seconds

Literal	Data type	Explanation
INTERVAL '-10' MINUTE	INTERVAL MINUTE	Minus 10 minutes.

4.6 Periods

A *period* is an object associated with a single base table. A period definition for a given table associates a period name with a pair of column names defined for that table. It is required that the columns are both of a datetime data type and known not nullable. Furthermore, it is required that the declared types of both columns are identical.

Similar to column definitions and constraint definitions, a period definition can only be specified as part of a table definition. For a table with a period definition, every row in that table is considered to be associated with a period whose name corresponds to the period name specified in the period definition and whose start and end times are provided by the column values specified in the period definition.

For example, consider the table definition:

```
CREATE TABLE emp
  (emp_id INTEGER NOT NULL,
   name VARCHAR(30),
   salary DECIMAL(5,2),
   dept_id INTEGER,
   bus_start DATE NOT NULL,
   bus_end DATE NOT NULL,
   PERIOD FOR business_time (bus_start, bus_end)
  );
```

The period definition “PERIOD FOR business_time (bus_start, bus_end)” in the preceding table definition defines a period named business_time for the emp table with the value in the bus_start column for a given row acting as the start time of the business_time period associated with the row and the value in the bus_end column for a given row acting as the end time of the business_time period associated with the row.

In general, for a period with name P, the first column in the period definition is called the P period start column, and the second column is called the P period end column. The columns participating in a period definition shall satisfy the conditions:

- 1) Both columns shall be declared as NOT NULL.
- 2) The data type of both columns shall be a datetime data type.
- 3) The data types of both columns shall be identical.

For any given period, the SQL-implementation ensures the value of the period end column is always greater than the value of the period start column. In general, the period is a set of datetime values consisting of every distinct value in the timeline starting from the period start value up to but not including the period end value.

A given table can have at most two period definitions. One of the periods is reserved for supporting system-time dimension, i.e., system-versioned tables with a predefined period name of “SYSTEM_TIME” for such a period. This leaves one period for supporting the application-time dimension with a user-defined name for such a period.

4.7 Operations involving datetimes and intervals

Table 9, “Valid operators involving datetimes and intervals”, specifies the declared types of arithmetic expressions involving datetime and interval operands.

Table 9 — Valid operators involving datetimes and intervals

Operand 1	Operator	Operand 2	Result Type
Datetime	-	Datetime	Interval
Datetime	+ or -	Interval	Datetime
Interval	+	Datetime	Datetime
Interval	+ or -	Interval	Interval
Interval	* or /	Numeric	Interval
Numeric	*	Interval	Interval

Arithmetic operations involving values of type datetime or interval obey the natural rules associated with dates and times and yield valid datetime or interval results according to the Gregorian calendar.

Operations involving values of type datetime require that the datetime values be comparable. Operations involving values of type interval require that the interval values be comparable.

Operations involving a datetime and an interval preserve the time zone of the datetime operand. If the datetime operand does not include a time zone displacement, then the result has no time zone displacement.

An extract expression operates on a datetime or interval and returns an exact numeric value representing the value of one component of the datetime or interval.

An interval absolute value function operates on an interval argument and returns its absolute value in the same most specific type.

4.8 Time-related predicates

4.8.1 Overlaps predicate

An overlaps predicate uses the operator OVERLAPS to determine whether or not two chronological periods overlap in time. A chronological period is specified either as a pair of datetimes (starting and ending) or as a starting datetime and an interval. If the length of the period is greater than 0 (zero), then the period consists of all points of time greater than or equal to the lower endpoint, and less than the upper endpoint. If the length of the period is equal to 0 (zero), then the period consists of a single point in time, the lower endpoint. Two chronological periods overlap if they have at least one point in common.

4.8.2 Period predicates

There are seven period predicates available, which are discussed in the list appearing later in this Sub-clause.

These predicates take two operands separated by keywords such as CONTAINS, OVERLAPS, etc. For predicates other than the period contains predicate, each of the operands can be either a period name or the syntactic construct of the form PERIOD (datetime value expression, datetime value expression), called a period constructor. For the period contains predicate, the first operand can be either a period name or a period constructor while the second operand can be either a period name, a period constructor, or a datetime value expression.

Note that the predicates can be used wherever predicate syntax is allowed, e.g., in the WHERE clause, on the ON clause of a joined table, etc.

A description of each of the predicates is provided in the following list (assume x is the first operand and y is the second operand; further assume that when x and y stand for periods, they are modelled as (closed, open) periods with xs and xe as the start and end times of period x and ys and ye as the start and end times of period y):

For the examples consider the table definition:

```
CREATE TABLE emp
  (emp_id INTEGER NOT NULL,
   name VARCHAR(30),
   salary DECIMAL(5,2),
   dept_id INTEGER,
   bus_start DATE NOT NULL,
   bus_end DATE NOT NULL,
   PERIOD FOR business_time (bus_start, bus_end)
);
```

- 1) The predicate “x OVERLAPS y” applies when both x and y are either period names or period constructors. This predicate returns *True* if the two periods have at least one time point in common, i.e., if $xs < ye$ and $xe > ys$.

A query to retrieve all emp rows whose application-time periods overlap a given period, say a period with the start date of 2001-01-01 and the end date of 2001-07-27 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp e
WHERE e.bus_start < DATE '2001-07-27' AND e.bus_end > DATE '2001-01-01'
```

The immediately preceding query can be written in a more succinct and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp e
WHERE e.business_time OVERLAPS PERIOD (DATE '2001-01-01', DATE '2001-07-27')
```

- 2) The predicate “x EQUALS y” applies when both x and y are either period names or period constructors. This predicate returns *True* if the two periods have every time point in common, i.e., if $xs = ys$ and $xe = ye$

A query to retrieve all emp rows whose application-time periods is equal to a given period, say a period with the start date of 2001-01-01 and the end date of 2001-07-27 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp e
WHERE e.bus_start = DATE '2001-01-01' AND e.bus_end = DATE '2001-07-27'
```

The immediately preceding example can be written in a more succinct and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp e
WHERE e.business_time EQUALS PERIOD (DATE '2001-01-01', DATE '2001-07-27')
```

3) The predicate “x CONTAINS y” applies when

- Both x and y are either period names or period constructors. In this case, the predicate returns *True* if x contains every time point in y, i.e., if $xs \leq ys$ and $xe \geq ye$.
- x is either a period name or a period constructor and y is a datetime value expression. In this case, the predicate returns *True* if x contains y, i.e., if $xs \leq y$ and $xe > y$.

A query to retrieve all emp rows whose application-time periods contain the date 2001-01-01 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp
WHERE bus_start <= DATE '2001-01-01' AND bus_end > DATE '2001-01-01'
```

The immediately preceding example can be written in a more succinct and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp
WHERE business_time CONTAINS DATE '2001-01-01'
```

4) The predicate “x PRECEDES y” applies when both x and y are either period names or period constructors. In this case, the predicate returns *True* if the end value of x is less than or equal to the start value of y, i.e., if $xe \leq ys$.

A query to retrieve all emp rows whose application-time periods precedes a given period, say a period with the start date of 2001-01-01 and the end date of 2001-07-27 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp e
WHERE e.bus_end <= DATE '2001-01-01'
```

The immediately preceding example can be written in a more explicit and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp e
WHERE e.business_time PRECEDES PERIOD (DATE '2001-01-01', DATE '2001-07-27')
```

5) The predicate “x SUCCEEDS y” applies when both x and y are either period names or period constructors. In this case, the predicate returns *True* if the start value of x is greater than or equal to the end value of y, i.e., if $xs \geq ye$.

A query to retrieve all emp rows whose application-time periods succeeds a given period, say a period with the start date of 2001-01-01 and the end date of 2001-07-27 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp e
WHERE e.bus_start >= DATE '2001-07-07'
```

The immediately preceding example can be written in a more explicit and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp e
WHERE e.business_time SUCCEEDS PERIOD (DATE '2001-01-01', DATE '2001-07-27')
```

6) The predicate “x IMMEDIATELY PRECEDES y” applies when both x and y are either period names or period constructors. In this case, the predicate returns *True* if the end value of x is equal to the start value of y, i.e., if $x_e = y_s$.

A query to retrieve all emp rows whose application-time periods immediately precedes a given period, say a period with the start date of 2001-01-01 and the end date of 2001-07-27 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp e
WHERE e.bus_end = DATE '2001-01-01'
```

The immediately preceding example can be written in a more explicit and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp e
WHERE e.business_time IMMEDIATELY PRECEDES
    PERIOD (DATE '2001-01-01', DATE '2001-07-27')
```

7) The predicate “x IMMEDIATELY SUCCEEDS y” applies when both x and y are either period names or period constructors. In this case, the predicate returns *True* if the start value of x is equal to the end value of y, i.e., if $x_s = y_e$.

A query to retrieve all emp rows whose application-time periods immediately succeeds a given period, say a period with the start date of 2001-01-01 and the end date of 2001-07-27 using existing SQL syntax looks as follows:

```
SELECT *
FROM emp e
WHERE e.bus_start = DATE '2001-07-07'
```

The immediately preceding example can be written in a more explicit and more intuitive way when the predicates in the WHERE clause are expressed using the name of the period rather than the start and end columns of the period, as shown here:

```
SELECT *
FROM emp e
WHERE e.business_time IMMEDIATELY SUCCEEDS
    PERIOD (DATE '2001-01-01', DATE '2001-07-27')
```

5 Time-related tables

5.1 Introduction to time-related tables

There are three different *flavors* of time-related tables. The first type has just an application-time period, the second has just the system versioning period, whereas the third type, called a bi-temporal table, has both types of periods.

5.2 Application-time period tables

5.2.1 Introduction to application-time period tables

Application-time period tables are intended for meeting the requirements of applications that are interested in capturing time periods during which the data is believed to be valid in the real world. A typical example of such applications is an insurance application, where it is necessary to keep track of the specific policy details of a given customer that are in effect at any given point in time.

A primary requirement of such applications is that the user be put in charge of setting the start and end times of the validity period of rows, and the user be free to assign any time values, either in the past, current or in the future, for the start and end times. Another requirement of such applications is that the user be permitted to update the validity periods of the rows as errors are discovered or new information is made available.

Any table that contains a period definition with a user-defined name is an application-time period table.

Users can pick any name they want for the name of the period as well as for the names of columns that act as the start and end columns of the period. The data types of the period start and end columns shall be either DATE or a timestamp type, and the data types of both columns shall be the same.

5.2.2 Extensions to primary key / unique constraints

Users can define primary key/unique constraints on tables containing an application-time period using the current syntax with exactly the same behavior. However, the presence of an application-time period provides an opportunity to enhance the notion of primary key/unique constraints on that table. For example, assume that an application requires the creation a primary key for the emp table that corresponds to the combination of the emp_id, bus_start, and bus_end columns, such that for a given emp_id value and a given point in time T, there is exactly one row whose application-time period (i.e., set of values from bus_start value through to but not including bus_end value) contains T. This means that for any selection based on a specified emp_id value and a specified date, one and only one row is retrieved. Assume the emp table contains the rows shown in [Table 10, “Example data table emp for primary key with application-time period”](#):

Table 10 — Example data table emp for primary key with application-time period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2002-01-01
100	Tom	3500	10	2002-01-01	2003-01-01
100	Tom	4000	20	2003-01-01	2004-01-01

In this example there are three rows. Note first, the application-time periods across all three rows are not overlapping. Note second, the end of the time period from row 1 is the start time from row 2, and the end time for row 2 is the start time of row 3. Note third that during each of these time periods, the employee is assigned to a different department and had a different salary. Note fourth that even though there are multiple rows for the same employee spanning a period from 2001-07-27 through 2004-01-01, there is only one emp row whose application-time period contains any specified date in the period from 2001-07-27 through 2004-01-01. For example, a specified date of 2002-12-01 would result in the selection of row 2. Suppose however, that row 2 is deleted. This would cause a gap in the period from 2001-07-27 through 2004-01-01. Thus, if there was a query to produce the dept_id and salary for the employee on the date, 2002-12-01, the query would fail because there is no row whose application-time period contains the date 2002-12-01. However, if there was a query to produce the dept_id and salary for the employee on the date, 2001-12-01 or on the date, 2003-12-01, it would still return only one employee row.

From the immediately preceding example, it is clear that a capability is needed to specify the emp table contains no two rows with the same emp_id value and overlapping application-time periods. There is additional syntax for primary key/unique constraint declarations to provide such a capability. This example illustrates this new syntax:

```
CREATE TABLE emp
  (emp_id INTEGER NOT NULL,
   name VARCHAR(30),
   salary DECIMAL(5,2),
   dept_id INTEGER,
   bus_start DATE NOT NULL,
   bus_end DATE NOT NULL,
   PERIOD FOR business_time (bus_start, bus_end),
   PRIMARY KEY (emp_id, business_time WITHOUT OVERLAPS)
  );
```

The PRIMARY KEY constraint in the immediately preceding definition ensures the table has no two rows with the same emp_id value with overlapping application-time periods. The reference to the business_time in the PRIMARY KEY declaration effectively makes the bus_start and bus_end columns as part of the primary key with check for overlapping periods rather than for equality of periods. For this example, an attempt to insert a new row with an emp_id value of 100 and application-time period from 2004-01-01 to 2005-01-01 will succeed while an attempt to insert a new row with an emp_id value of 100 and application-time period from 2003-06-01 to 2004-06-01 will fail.

5.2.3 Extensions to referential constraints

Currently, a referential constraint between a referencing table and a referenced table ensures, for every row R in the referencing table, values in the foreign key columns of R match with the values of the primary/unique key columns of exactly one row in the referenced table. However, the presence of an

5.2 Application-time period tables

application-time period in both referencing and referenced tables provides an opportunity to enhance the notion of referential constraints between those tables. For example, creation of a referential constraint between the emp table as defined in [Subclause 5.2.2, “Extensions to primary key / unique constraints”](#), and the dept table shown here is required:

```
CREATE TABLE dept
(dept_id INTEGER NOT NULL,
 name VARCHAR(30),
 budget DECIMAL(5,2),
 bus_start DATE NOT NULL,
 bus_end DATE NOT NULL,
 PERIOD FOR business_time (bus_start, bus_end),
 PRIMARY KEY (dept_id, business_time WITHOUT OVERLAPS)
);
```

Assume that the dept table has a primary key that corresponds to the combination of the dept_id column and the business_time period as shown in the immediately preceding table definition. Assume further that creation of a referential constraint is required between the emp table and the dept table with the emp table's foreign key that corresponds to the combination of the dept_id column and the business_time period, such that for a given row R in emp table, for every point in time T in R's application-time period (i.e., set of values from bus_start value through to but not including bus_end value), there is exactly one matching row in the dept table whose application-time period contains T.

Assume the contents of dept and emp tables are as shown in [Table 11, “Example data table dept for foreign key with application-time period”](#), and [Table 12, “Example data table emp for foreign key with application-time period”](#), respectively:

Table 11 — Example data table dept for foreign key with application-time period

dept_id	name	budget	bus_start	bus_end
1	Server	30000	2000-03-01	2002-01-01
1	Server	35000	2002-01-01	2003-01-01
2	Tools	40000	2003-01-01	2004-01-01

Table 12 — Example data table emp for foreign key with application-time period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2002-07-27
100	Tom	3500	1	2002-07-27	2003-01-01
100	Tom	4000	2	2003-01-01	2003-06-01

Examining the content of each table, it is clear that the dept_id value of both row 1 and row 2 of emp table (i.e., 1) matches with the dept_id value of both row 1 and row 2 of dept table and the dept_id value of row 3 of emp table (i.e., 2) matches with the dept_id value of exactly one row, row 3, of dept table. It is also clear that the application-time period of row 1 of emp table is not completely contained in the application-time period of either row 1 or row 2 of dept table since even though the bus_start value of row 1 of emp table occurs after the bus_start value of row 1 of dept table, the bus_end value of row 1 of emp table occurs after the bus_end value of row 1 of dept table and even though the bus_end value of row 1 of emp table occurs before the bus_end value of row 2 of dept table, the bus_start value of row 1

of emp table occurs before the bus_start value of row 2 of dept table. For row 2 of emp table, it is evident that its application-time period of is completely contained in the application-time period of row 2 of dept table. Similarly, it is apparent that the application-time period of row 3 of emp table is completely contained in the application-time period of row 3 of dept table.

The fact that the dept_id value of a given row (say row 1 of emp table) matches with the dept_id value of more than one row of dept table may seem like a potential violation of referential integrity between emp table and the dept table. However, that is really not the case. This is because for every date value in the application-time period of row 1 of emp table, i.e., for every date value starting from 2001-07-27 through but not including 2002-07-27, the dept_id value of row 1 of emp table matches with the dept_id value of exactly one row (either row 1 or row 2) of dept table. For instance, for a specific date such as 2001-08-30, the dept_id value of row 1 of emp table matches with the dept_id value of row 1 of dept table. For another date such as 2002-03-01, the dept_id value of row 1 of emp table matches with the dept_id value of row 2 of dept table. This is true for every date value starting from 2001-07-27 through 2002-07-26. It is the same case for every date value in the application-time period of row 2 of emp table and for every date value in the application-time period of row 3 of emp table.

From the immediately preceding example, it is clear that capability is needed to specify for every point in time T in the application-time period of a given row R in emp table, there is exactly one matching row in the dept table whose application-time period contains T. There is additional syntax for referential constraint declarations to provide such a capability. This example illustrates this new syntax:

```
CREATE TABLE emp
  (emp_id INTEGER NOT NULL,
   name VARCHAR(30),
   salary DECIMAL(5,2),
   dept_id INTEGER,
   bus_start DATE NOT NULL,
   bus_end DATE NOT NULL,
   PERIOD FOR business_time (bus_start, bus_end),
   PRIMARY KEY (emp_id, business_time WITHOUT OVERLAPS),
   FOREIGN KEY (dept_id, PERIOD business_time) REFERENCES
     dept (dept_id, PERIOD business_time)
);
```

In the immediately preceding example, the reference to the period name in the FOREIGN KEY clause as well as in the REFERENCES clause is prefixed with the keyword PERIOD to indicate the special checking that needs to be done to enforce referential integrity. For this example, an attempt to insert a new row with an emp_id value of 100, dept_id value of 1, and application-time period from 2000-06-01 to 2001-01-01 into emp table will succeed while an attempt to insert a new row with an emp_id value of 100, dept_id value of 1, and application-time period from 2000-01-01 to 2001-01-01 will fail.

Note that in the immediately preceding example, both the referencing and referenced tables contain an application-time period. If only one of the tables contains an application-time period and the other does not, then the only kind of referential constraint that can be defined between them is a regular referential constraint, i.e., a referential constraint that does not involve periods. For example, consider these two scenarios:

- 1) Referenced table contains an application-time period but the referencing table does not: If the primary/unique key constraint on the referenced table includes an application-time period, then it is not possible to define a referential constraint involving periods between such a referenced table and referencing table since there is no way to associate a period with the referencing table's foreign key columns. On the other hand, if the primary/unique key constraint on the referenced table does not include the application-time period, then it is perfectly legal to define a regular referential constraint between such a referenced table and referencing table. However, such a situation is hardly likely to occur since the primary/unique constraint on a referenced table that includes an application-time period is likely to include the application-time period in its definition in almost all cases.

2) Referenced table does not contain an application-time period but the referencing table does: Since there is no way for the primary/unique constraint on the referenced table to include an application-time period, there is no way to define a referential constraint involving periods in this scenario as well. On the other hand, it is always possible to define a regular referential constraint between such a referenced table and referencing table.

5.2.4 Inserting rows into tables containing an application-time period definition

Rows can be inserted into tables containing an application-time period in exactly the same way as inserting rows into any tables. The only difference is that there is an automatically-generated non-deferrable constraint for such tables that ensures that the value of the end column of the application-time period is greater than the value of the start column of the application-time period for every row being inserted. An exception would be raised if that is not the case. For example, assume the emp table was empty when this INSERT statement:

```
INSERT INTO emp
VALUES (100, 'Tom', 3000, 1, DATE '2001-07-27', DATE '2004-07-27');
```

is executed. The content of emp table is changed as shown in Table 13, "Content of table emp after insert with application-time period":

Table 13 — Content of table emp after insert with application-time period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

The execution of the INSERT statement shown before Table 13, "Content of table emp after insert with application-time period", will activate the INSERT triggers defined on emp table in the usual way.

5.2.5 Updating rows of tables containing an application-time period definition

Update operations on tables containing an application-time period apply specified updates to all qualifying rows, exactly like the update operations on a regular table. For example, if the emp table contains the rows shown in Table 14, "Content of table emp before updating a row":

Table 14 — Content of table emp before updating a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27
100	Tom	3000	2	2004-07-27	2006-07-27

the UPDATE statement:

```
UPDATE emp
SET dept_id = 10
WHERE emp_id = 100;
```

modifies the content of emp table as shown in Table 15, "Content of table emp after updating a row":

Table 15 — Content of table emp after updating a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	10	2001-07-27	2004-07-27
100	Tom	3000	10	2004-07-27	2006-07-27

Users can also update the start column and/or the end column of a period, as shown in this example.

```
UPDATE emp
SET dept_id = 10,
    bus_start = DATE '2002-07-27'
WHERE emp_id = 100;
```

Assuming the content of emp table before the execution of the immediately preceding example was as shown in [Table 16, “Content of table emp before updating the application-time period of a row”](#):

Table 16 — Content of table emp before updating the application-time period of a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

The content of emp table after the execution is shown in [Table 17, “Content of table emp after updating the application-time period of a row”](#):

Table 17 — Content of table emp after updating the application-time period of a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	10	2002-07-27	2004-07-27

Whenever the start and/or the end column of an application-time period is updated, a check is made to make sure that the value specified for the end column of the application-time period is greater than the value specified for the start column of the application-time period after every row is updated. An exception would be raised if that is not the case. All UPDATE triggers defined on the table will get activated in the usual way for all rows being updated.

5.2.6 Updating the table between specific points in time

For tables containing an application-time period, the presence of the application-time period provides an opportunity to enhance the behavior of UPDATE statement. For example, users may want to specify a time period as part of the UPDATE statement such that their specified updates apply only to those rows whose application-time periods either overlap or are contained in the specified period. Additionally, they may want the specified updates to apply in the usual way to those rows whose application-time periods are completely contained in the specified period, but for rows whose application-time periods overlap the specified period, they may want the specified updates to apply only for the intersecting period without losing the old content of rows for the non-intersecting period. There is additional syntax for UPDATE statement that allows exactly this capability. This example illustrates this new syntax:

```

UPDATE emp FOR PORTION OF business_time
  FROM DATE '2002-01-01' TO DATE '2003-01-01'
SET dept_id = 10
WHERE emp_id = 100;

```

Assume the content of emp table prior to the execution of the immediately preceding example was as shown in [Table 18](#), “Content of table emp before updating a row for an inner part of a period”:

Table 18 — Content of table emp before updating a row for an inner part of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

Assuming successful execution of the previous UPDATE statement, the content of emp table is changed as shown in [Table 19](#), “Content of table emp after updating a row for an inner part of a period”:

Table 19 — Content of table emp after updating a row for an inner part of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2002-01-01
100	Tom	3000	10	2002-01-01	2003-01-01
100	Tom	3000	1	2003-01-01	2004-07-27

As a result of the preceding UPDATE statement, the existing row for Tom is updated to show that he is assigned to department 10 from 2002-01-01 to 2003-01-01 while two new rows are inserted that show that Tom is assigned to department 1 from 2001-07-27 to 2002-01-01 and from 2003-01-01 to 2004-07-27.

If an UPDATE statement contains a FOR PORTION OF clause, explicit updates to either the start column or the end column of the application-time period are not allowed.

This example illustrates the case where the candidate row’s application-time period is contained in the period specified by the FROM and TO values specified in the FOR PORTION OF clause, and hence no new rows are inserted:

```

UPDATE emp FOR PORTION OF business_time
  FROM DATE '2001-01-21' TO DATE '2004-12-31'
SET dept_id = 10
WHERE emp_id = 100;

```

Assume the content of emp table prior to the execution of above statement was as shown in [Table 20](#), “Content of table emp before updating a row for all of a period”:

Table 20 — Content of table emp before updating a row for all of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

Assuming successful execution of the preceding UPDATE statement, the content of emp table is changed as shown in Table 21, “Content of table emp after updating a row for all of a period”:

Table 21 — Content of table emp after updating a row for all of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	10	2001-07-27	2004-07-27

Here is an example where only one new row is inserted per row update:

```
UPDATE emp FOR PORTION OF business_time
  FROM DATE '2001-07-27' TO DATE '2003-01-01'
SET dept_id = 10
WHERE emp_id = 100;
```

Assume the content of emp table prior to the execution of immediately preceding example was as shown in Table 22, “Content of table emp before updating the edge of a period of a row”:

Table 22 — Content of table emp before updating the edge of a period of a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

Assuming successful execution of the preceding UPDATE statement, the content of emp table is changed as shown in Table 23, “Content of table emp after updating the edge of a period of a row”:

Table 23 — Content of table emp after updating the edge of a period of a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	10	2001-07-27	2003-01-01
100	Tom	3000	1	2003-01-01	2004-07-27

More precisely, the effect of UPDATE statements that contain the FOR PORTION OF clause are as follows:

- 1) Let FT be the first value and ST be the second value specified in the FOR PORTION OF clause.
- 2) For each row R in the table that qualifies for update and whose application-time period overlaps with the period formed by FT and ST, let BPS be its application-time period start value, and let BPE be its application-time period end value.
 - a) If BPS < FT and BPE > FT, then a copy of R with its application-time period end value set to FT is inserted.
 - b) If BPS < ST and BPE > ST, then a copy of R with its application-time period start value set to ST is inserted.
 - c) R is updated with its application-period start value set to the maximum of BPS and FT and the application-time end value set to the minimum of BPE and ST.

All UPDATE triggers defined on the table will get activated in the usual way for all rows that are updated. In addition, all INSERT triggers will get activated for all rows that are inserted. Currently it is not possible

for the body of an UPDATE trigger to gain access to the FROM and TO values in the FOR PORTION OF clause if one is specified.

5.2.7 Deleting rows from tables containing an application-time period definition

Delete operations on tables containing an application-time period delete all qualifying rows, exactly like the delete operations on a regular table. For example, the delete operation:

```
DELETE FROM emp
WHERE emp_id = 100;
```

deletes all rows with emp_id value of 100. All DELETE triggers defined on the table will get activated for each row being updated in the usual way.

5.2.8 Deleting rows between specific points in time

As in the case of update operations, for tables containing an application-time period, the presence of application-time period provides an opportunity to enhance the behavior of DELETE statement. For example, users may want to specify a time period as part of the DELETE statement such that the delete operation applies only to those rows whose application-time periods either overlap or are contained in the specified period. Additionally, they may want the delete to apply in the usual way to those rows whose application-time periods are completely contained in the specified period, but for rows whose application-time periods overlap the specified period, they may want the delete to apply only for the intersecting period without losing the old content of rows for the non-intersecting period. To provide this functionality, there is additional syntax for DELETE statement that mirrors the additional syntax created for the UPDATE statement.

This example illustrates this new syntax:

```
DELETE FROM emp FOR PORTION OF business_time
          FROM DATE '2002-01-01' TO DATE '2003-01-01'
WHERE emp_id = 100;
```

Assume the content of emp table prior to the execution of the immediately preceding example was as shown in [Table 24, “Content of table emp before deleting an inner part of a period”](#):

Table 24—Content of table emp before deleting an inner part of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

Assuming successful execution of the preceding DELETE statement, the content of emp table is changed as shown in [Table 25, “Content of table emp with application-time period after deleting an inner part of a period”](#):

Table 25 — Content of table emp with application-time period after deleting an inner part of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2002-01-01
100	Tom	3000	1	2003-01-01	2004-07-27

Note that as a result of the preceding DELETE statement, the existing row for Tom is deleted and two copies of the existing row for Tom showing he is assigned to department 1 from 2001-07-27 to 2002-01-01 and from 2003-01-01 to 2004-07-27 are inserted.

This example illustrates the case where the candidate row's application-time period is contained in the period specified by the FROM and TO values specified in the FOR PORTION OF clause, and hence no new rows are inserted:

```
DELETE FROM emp FOR PORTION OF business_time
  FROM DATE '2001-01-01' TO DATE '2004-12-31'
 WHERE emp_id = 100;
```

Assume the content of emp table prior to the execution of the immediately preceding example was as shown in [Table 26, “Content of table emp with application-time period before deleting a row”](#):

Table 26 — Content of table emp with application-time period before deleting a row

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

Assuming successful execution of the preceding DELETE statement, the content of emp table will no longer contain a row for Tom; that is, the table will be empty.

Here is an example where only one new row is inserted per row delete:

```
DELETE FROM emp FOR PORTION OF business_time
  FROM DATE '2001-07-27' TO DATE '2003-07-27'
 WHERE emp_id = 100;
```

Assume the content of emp table prior to the execution of above statement was as shown in [Table 27, “Content of table emp before deleting an edge of a period”](#):

Table 27 — Content of table emp before deleting an edge of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2001-07-27	2004-07-27

Assuming successful execution of the DELETE statement in the immediately preceding example, the content of emp table is changed as shown in [Table 28, “Content of table emp after deleting an edge of a period”](#):

Table 28 — Content of table emp after deleting an edge of a period

emp_id	name	salary	dept_id	bus_start	bus_end
100	Tom	3000	1	2003-07-27	2004-07-27

More precisely, the effect of DELETE statements that contain the FOR PORTION OF clause are as follows:

- 1) Let FT be the first value and ST be the second value specified in the FOR PORTION OF clause.
- 2) For each row R in the table that qualifies for deletion and whose application-time period overlaps with the period formed by FT and ST, let BPS be its application-time period start value, and let BPE be its application-time period end value.
 - a) If BPS < FT and BPE > FT, then a copy of R with its application-time period end value set to FT is inserted.
 - b) If BPS < ST and BPE > ST, then a copy of R with its application-time period start value set to ST is inserted.
 - c) R is deleted.

All DELETE triggers defined on the table will get activated in the usual way for all rows that are deleted. In addition, all INSERT triggers will get activated for all rows that are inserted. As in the case of UPDATE triggers, it is currently not possible for the body of a DELETE trigger to gain access to the FROM and TO values in the FOR PORTION OF clause if one is specified.

5.2.9 Querying tables containing a period definition

The standard does not provide any extensions to the SQL query syntax that are specifically targeted to querying tables containing an application-time period. This is because there is a tremendous body of research (and several competing research proposals) on temporal query extensions and the job of coming up with a simple yet complete query proposal for tables containing application-time period definitions proved to be more challenging than expected.

NOTE 3 — There are currently no plans to add any such extensions to ISO/IEC 9075-2.

Users can, however, use the existing query syntax for querying tables containing an application-time period in exactly the same way as for querying regular tables. For example, this query retrieves rows whose application-time periods intersect a specified point in time:

```
SELECT *
FROM emp
WHERE bus_start ≤ DATE '2001-01-01'
AND bus_end > DATE '2001-01-01';
```

Similarly, the next query retrieves rows whose application-time periods overlap with the period formed by 2001-01-01 and 2001-07-07 (assuming the closed-open model of period values):

```
SELECT *
FROM emp
WHERE bus_start < DATE '2001-07-27'
AND bus_end > DATE '2001-01-01';
```

Application-time period tables can be queried using the regular query syntax. For example, to retrieve the department where the employee 22217 worked as of January 2, 2011, one can express the query as:

```
SELECT Name, Edept
FROM Emp
WHERE ENo = 22217
AND EStart ≤ DATE '2011-01-02'
AND EEnd > DATE '2011-01-02'
```

A simpler way to formulate the immediately preceding example would be to employ one of the period predicates explained in Subclause 4.8.2, “Period predicates” for expressing conditions involving periods: CONTAINS, OVERLAPS, EQUALS, PRECEDES, SUCCEEDS, IMMEDIATELY PRECEDES, and IMMEDIATELY SUCCEEDS. For example, the immediately preceding example could also be expressed using the CONTAINS predicate:

```
SELECT Ename, Edept
FROM Emp
WHERE ENo = 22217 AND
EPeriod CONTAINS DATE '2011-01-02'
```

If one wanted to know all the departments where the employee whose number is 22217 worked during the period from January 1, 2010 to January 1, 2011, one could formulate the query as:

```
SELECT Ename, Edept
FROM Emp
WHERE ENo = 22217
AND EStart < DATE '2011-01-01'
AND EEnd > DATE '2010-01-01'
```

Note that the period specified in the immediately preceding example uses the closed-open model, i.e., the period includes January 1, 2010 but excludes January 1, 2011. Alternatively, the same query could be expressed using the OVERLAPS predicate as:

```
SELECT Ename, Edept
FROM Emp
WHERE ENo = 22217
AND EPeriod OVERLAPS PERIOD (DATE '2010-01-01', DATE '2011-01-01')
```

5.2.10 Adding a period definition to a table

The ALTER TABLE statement is extended with two new options extended, ADD PERIOD and DROP PERIOD, to allow for adding an application-time period to an existing table that does not have such a period and for dropping an application-time period from a table that does have such a period. For example, assume this table exists in some schema:

```
CREATE TABLE dept
(dep_id INTEGER NOT NULL,
 dep_name VARCHAR(30),
 dept_budget DECIMAL(10, 2),
 bus_start DATE NOT NULL,
 bus_end DATE NOT NULL
);
```

This ALTER TABLE statement adds the application-time period to the dept table:

```
ALTER TABLE dept ADD PERIOD FOR business_time (bus_start, bus_end);
```

When the immediately preceding example executes, a check is made to ensure that both bus_start and bus_end columns have a non-deferrable NOT NULL constraints defined on them and to ensure the value of bus_end column is greater than the value of bus_start column for all existing rows in dept table; the

execution fails if the check fails. A non-deferrable CHECK constraint is implicitly added to the table to ensure the value of bus_end column is greater than the value of bus_start column at the end of every DML operation.

This ALTER TABLE statement drops the application-time period from the dept table:

```
ALTER TABLE dept DROP PERIOD FOR business_time RESTRICT;
```

or via:

```
ALTER TABLE dept DROP PERIOD FOR business_time CASCADE;
```

Dropping the application-time period from a table impacts the constraints, triggers, views, and routines defined on the table if those objects contain references to the application-time period.

For a table containing an application-time period table, it is prohibited to drop the NOT NULL constraints on the application-period start and end columns and the implicit CHECK constraint added to make sure the value of application-time end column is greater than the value of application-time start column. To drop these constraints, users could either drop the application-time period before attempting to drop the constraints or drop the period under CASCADE.

5.3 System-versioned tables

5.3.1 Introduction to system-versioned tables

System-versioned tables are intended for meeting the requirements of applications that must maintain an accurate history of data changes either for business reasons, legal reasons, or both. A typical example of such applications is a banking application, where it is necessary to keep previous states of customer account information so that customers can be provided with a detailed history of their accounts. There are also plenty of examples where certain institutions are required by law to preserve historical data for a specified length of time to meet regulatory and compliance requirements.

A key requirement of such applications is that any update or delete of a row automatically preserves the old state of the row before performing the update or delete. Another important requirement is that the system, rather than the user, maintains the start and end times of the periods of the rows, and that users be unable to modify the content of historical rows or the periods associated with any of the rows. Any updates to the periods of rows in a system-versioned table shall be performed only by the system as a result of updates to the non-period columns of the table or as a result of row deletions. This provides the guarantee that the recorded history of data changes cannot be tampered with, which is critical to meet auditing and compliance regulations.

Any table that contains a period definition with the standard-specified name, SYSTEM_TIME, and includes the keywords WITH SYSTEM VERSIONING in its definition is a system-versioned table. Similar to application-time period tables, users can pick any name they want for the names of columns that act as the start and end columns of the SYSTEM_TIME period. Though ISO/IEC 9075-2 allows the data types of the period start and end columns to be either DATE or a timestamp type (as long as the data types of both columns are the same), in practice, most SQL-implementations will provide the TIMESTAMP type with the highest fractional seconds precision as the data type for the system-time period start and end columns. For example:

```
CREATE TABLE Emp
(ENO INTEGER,
 Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
 Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
```

```

ENAME VARCHAR(30),
PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end)
) WITH SYSTEM VERSIONING

```

Similar to application-time periods, system-time periods use closed-open period model. At any given point in time, a row in a system-versioned table is regarded as current system row if the system-time period of that row contains the current time. A row that is not a current system row is regarded as a historical system row.

System-versioned tables differ from application-time period tables in these respects:

- 1) In contrast to the application-time period tables, users are not allowed to assign or change the values of Sys_start or Sys_end columns; they are assigned (and changed) automatically by the database system. This is the reason why the definitions of Sys_start or Sys_end columns shall include the keywords GENERATED ALWAYS.
- 2) INSERT into a system-versioned table automatically sets the value of Sys_start column to the transaction timestamp, a special value associated with every transaction, and sets the value of Sys_end column to the highest value of the column's data type.

ISO/IEC 9075-2 leaves it up to SQL-implementations to pick an appropriate value for the transaction timestamp of a transaction, but it does require the transaction timestamp of a transaction to remain fixed during the entire transaction.

- 3) UPDATE and DELETE on system-versioned tables only operate on current system rows. Users are not allowed to update or delete historical system rows. Users are also not allowed to modify the system-time period start or the end time of both current system rows and historical system rows.
- 4) UPDATE and DELETE on system-versioned tables result in the automatic insertion of a historical system row for every current system row that is updated or deleted.
- 5) <system time period start column specification> and <system time period end column specification> are defined using <column definition>s that *implicitly* include NOT NULL constraints.

An UPDATE statement on a system-versioned table first inserts a copy of the old row with its system-time period end time set to the transaction timestamp, indicating that the row ceased to be current as of the transaction timestamp. It then updates the row while changing its system-period start time to the transaction timestamp, indicating that the updated row to be the current system row as of the transaction timestamp.

5.3.2 Primary key and referential constraints

The definition and enforcement of constraints on system- versioned tables is considerably simpler than the definition and enforcement of constraints on application- time period tables. This is because constraints on system-versioned tables need only be enforced on the current system rows. Historical system rows in a system- versioned table form immutable snapshots of the past. Any constraints that were in effect when a historical system row was created would have already been checked when that row was a current system row, so there is never any need to enforce constraints on historical system rows. Consequently, there is no need to include the system-period start and end columns or the period name in the definition of primary key and referential constraints on system-versioned tables. For example, this ALTER TABLE statement specifies ENo column as the primary key of Emp table:

```

ALTER TABLE Emp
ADD PRIMARY KEY (ENO)

```

The constraint in the immediately preceding example ensures there exists exactly one current system row with a given ENo value.