ANSI X3H2-96-152
ISO/IEC JTC1/SC21/WG3 DBL ?

I S O
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

April 15, 1996

Subject: SQL/Temporal

Status: Change Proposal

Title: Adding Transaction Time to SQL/Temporal

Source: ANSI Expert's Contribution

Authors: Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen and Andreas Steiner

Abstract: Transaction time identifies when data was asserted in the database. If transaction time is supported, the states of the database at all previous points of time are retained. This change proposal specifies the addition of transaction time, in a fashion consistent with that already proposed for valid time. In particular, constructs to create tables with valid-time and transaction-time support and query such tables with temporal upward compatibility, sequenced semantics, and nonsequenced semantics, orthogonally for valid and transaction time, is defined. These constructs also can be used in modifications, assertions, cursors, and views.

References

[1] Melton, J. (ed.) *SQL/Foundation.* October, 1995. (ISO/IEC JTC 1/SC 21/WG 3 DBL-LHR-002.)

[2] Melton, J. (ed.) *SQL/Temporal.* October, 1995. (ISO/IEC JTC 1/SC 21/WG 3 DBL-LHR-009.)

[3] Snodgrass, R. T., M. H. Böhlen, C. S. Jensen and A. Steiner *Adding Valid Time to SQL/Temporal*, ANSI X3H2-96-151, March, 1996.

[4] Steiner, A. and M. H. Böhlen. The TimeDB Temporal Database Prototype, September, 1995. Available at `ftp://www.iesd.auc.dk/general/DBS/tdb/TimeCenter` or at `ftp://ftp.cs.arizona.edu/tsql/timecenter/TimeDB.tar.gz`.

1 Introduction

Transaction time identifies when data was asserted in the database. If transaction time is supported, the states of the database at all previous points of time are retained.

2 The Problem

Many databases need to keep track of the past state of the database, often for auditing requirements. Changes are not allowed on the past state; that would prevent secure auditing. Instead, compensating transactions are used to correct errors.

When an error is encountered, often the analyst will look at the state of the database at a previous point in time to determine where and how the error occurred.

However, SQL currently does not support such modifications or queries well. The following example will illustrate the problems.

- Assume that we wish to keep track of the changes and deletions of the Employee table discussed in the previous change proposal [3]. This table has four columns: Name, Manager, Dept, and When (a `PERIOD` indicating when the row was valid). To know when rows are inserted and (logically) deleted, we add two more columns, InsertTime and DeleteTime, both of the data type `TIMESTAMP`. Of course, adding these two columns breaks the referential integrity constraint between Manager and Name (the manager must also be an employee). The reader is invited to reformulate this referential integrity, taking into account the three time columns.
- We find out that the telephone bill for a department is unusually high, so we ask, "How many employees have been in each department," to get a start. This query is quite complex to formulate in SQL.
- It turns out that one of the departments shows an unreasonable number of current employees (more than 25). When was the error introduced? Is this inconsistency in the database widespread? How long has the database been incorrect? The query, "When did we think that departments are overly large?", provides an initial answer, but is also very difficult to express in SQL.

These queries are very challenging, even for SQL experts, when time is involved.

Modifications are even more of a problem. A logical deletion must be implemented as an update and an insertion, because we don't want to change the previously stored information. However, there is no way of preventing an application from inadvertently corrupting past states (by incorrectly altering the values of the InsertTime or DeleteTime columns), or a white collar criminal from intentionally "changing history" to cover up his tracks.

3 Outline of the Solution

The solution is to have the DBMS maintain transaction time automatically, so that the integrity of the previous states of the database is preserved. The query language can also help out, by making it easy to write queries.

With the small syntactic additions proposed here, transaction time can be easily added.

```
ALTER TABLE Employee ADD TRANSACTION
```

Since the DBMS is maintaining transaction time for us, for this table, we don't have to worry about the integrity of the previous states. The DBMS simply won't let us modify past states.

The previously specified sequenced valid referential integrity still applies, always on the current state of the database. No rephrasing of this integrity constraint is necessary.

The query, "How many employees have been in each department," asks for the history in valid time of the current transaction-time state. Hence, it is particularly easy to specify, by exploiting transaction-time upward compatibility.

```
VALID SELECT Dept, COUNT(*)
FROM Employee
GROUP BY Dept
```

To find where the error was made, we write the query, "When did we think that departments are overly large?" This uses the current time in valid time (the current departments), but looks at past states of the database. This requires a sequenced transaction query, with valid-time temporal upward compatibility.

```
TRANSACTION SELECT Dept, COUNT(*)
FROM Employee
GROUP BY Dept
HAVING COUNT(*) > 25
```

By having the DBMS maintain transaction time, applications that need to retain past states of tables for auditing purposes can have these past states maintained automatically, correctly, and securely. As well, the proposed language extensions enable queries to be written in minutes instead of hours.

4 Scope

This change proposal adds transaction-time support to SQL/Temporal. These facilities augment the valid-time support proposed earlier [3]. Transaction-time support provides the following features.

- Both valid-time and transaction-time support are optional.
- Tables with transaction-time support can be converted, via a view or within a query or cursor, to a conventional table with an additional period column, if the user prefers to manipulate the data in that fashion.
- Temporal upward compatible, sequenced, and nonsequenced queries can all be expressed on tables with valid-time and transaction-time support, orthogonally.

5 Transaction Time

As we saw in previous change proposal [3], valid time concerns the time when a fact is true in reality. The valid time of a fact is the wall clock time at which the fact was true in the modeled reality, independent of the recording of that fact in some database. Valid times can be in the future, if it is known that some fact will become true at a specified time in the future.

Orthogonally to valid time, transaction time can be associated with tables. The transaction time of a row, which is a period, specifies when that row was considered to be logically stored in the database. If the row (Tony, 10000, LeeAnn) was stored in the database on March 15, 1992 (say, with an `INSERT` statement) and removed from the database on June 1, 1992 (say, with a `DELETE` statement), then the transaction time of that row would be the period from March 15, 1992 to June 1, 1992.

The domain of transaction time does not extend past now (no facts are known to have been stored in the future). As the database state evolves, transaction times grow monotonically, and successive transactions have successive transaction times associated. In contrast, successive transactions may mention widely varying valid times.

Since transaction time is orthogonal to valid time, a table can have no temporal support, only valid-time support, only transaction-time support, and both valid- and transaction-time support.

EXAMPLE 1: Consider a table with both valid-time and transaction-time support recording employee information, such as "Jake works for the shipping department." We assume that the precision of the timestamps is one day for both valid time and transaction time (though in reality the precision of transaction time is probably a fraction of a second).

Figure 1 gives a sample table that illustrates Jake's interesting employment history. Jake was hired by the company as temporary help in the Shipping department for the interval from June 10 to June 15, and this fact became current in the database at June 5.

Later, the Personnel department discovers that Jake had really been hired from June 5 to June 20, and the database is corrected on June 10. Later, the Personnel department is informed that the correction was itself incorrect; Jake really was hired for the original time interval, June 10 to June 15, and the correction took effect in the database on June 15. Finally, on June 20, the Personnel department determines that, while the period of validity was correct, Jake was not in the Shipping department, but in the *Loading* department (!). Consequently, the fact (Jake, Shipping) is removed from the current state and the fact (Jake, Loading) is inserted. In the table, we represent the current time in transaction time with a NULL value internally. Alternatively, we could have represented it with a large timestamp like 9999-12-31. As we will see, users will never encounter transaction times greater than CURRENT_TIMESTAMP.

Emp	Dept	Valid Time	Transaction Time
Jake	Shipping	[1995-06-10 - 1995-06-16)	[1995-06-05 - 1995-06-10)
Jake	Shipping	[1995-06-05 - 1995-06-21)	[1995-06-10 - 1995-06-15)
Jake	Shipping	[1995-06-10 - 1995-06-16)	[1995-06-15 - 1995-06-20)
Jake	Loading	[1995-06-10 - 1995-06-16)	[1995-06-20 - NULL)

Figure 1: A Table With Both Valid-Time And Transaction-Time Support

With this table with both valid-time and transaction-time support, we can ask many interesting queries. Some queries take a vertical slice at a particular transaction time, determining what was recorded in the database at that time.

- As best known, who worked in the various departments?

  Jake worked in the Loading department.

- As recorded in the database in June 18, 1995 (perhaps erroneously), who worked in the various departments?

  Jake worked in the Shipping department.

- Rolling back the database to June 12, 1995, how long did we think Jake was scheduled to work?

  Jake was scheduled to work 16 days, from June 5 to June 20.

Other queries take a horizontal slice at a particular valid time.

- Concerning June 12, 1995, who worked then, as best known now?

  Jake worked in the Loading department then.

- What erroneous data was corrected concerning June 12, 1995?

  We thought Jake was working in the Shipping department on June 12 (this data was stored on June 5), but his department was corrected on June 20 to the Loading department.

□

The concepts of temporal upward compatibility *(TUC)*, sequenced *(SEQ)*, and nonsequenced *(NONSEQ)* semantics applies orthogonally to valid time and transaction time.

EXAMPLE 2: Assume that we have an *employee* table with attributes Name, Salary, and Manager. We can state queries that are different combinations of TUC, SEQ, and NONSEQ in valid and transaction time. In the following, we indicate valid time, then transaction time. Hence, "TUC/SEQ" means valid-time upward compatible and sequenced transaction-time semantics.

TUC/TUC Who currently makes more than their manager, as best known?

A table with no temporal support results.

SEQ/TUC Who at any time makes or made more than their manager did (at the same time, as best known)?

A table with valid-time support results.

TUC/SEQ Who did we think makes more than their manager today?

NONSEQ/TUC Who made more than their manager did (at any time), as best known?

A table with no temporal support results.

TUC/NONSEQ When was it recorded that someone currently makes more than their manager?

A table with no temporal support results.

SEQ/SEQ When did we think that someone, at some time, made more than their manager, at the same time?

A table with both valid-time and transaction-time support results.

SEQ/NONSEQ When did we correct the information to record that someone, at some time, made more than their manager, at the same time?

A table with valid-time support results. For each transaction time, we get a row with valid-time support, indicating when the employee is now considered to make more than their manager.

NONSEQ/SEQ Who was recorded, perhaps erroneously, to have made more than their manager did at any time?

Here we get a table with transaction-time support, indicating when the perhaps erroneous data was in the table.

NONSEQ/NONSEQ When did we correct the information, to record that someone made more than their manager did, at any time?

Here a table with no temporal support results.

TUC in valid time translates in English to "at now"; SEQ translates to "at the same time'; and NONSEQ translates to "at any time." TUC in transaction time translates to "as best known"; SEQ translates to "when did we think ... at the same time"; and NONSEQ translates to "when was it recorded that."

This example illustrates that all combinations are meaningful. □

While this example emphasized the orthogonally of valid and transaction time, that TUC, SEQ, and NONSEQ can be applied equally to both, there are still some differences between the two types of time.

First, valid time can have a precision specified by the user at table creation time. The transaction timestamps have an implementation-dependent range and precision. Second, valid time extends into the future, whereas transaction time always ends at now. Finally, during modifications the DBMS provides the transaction time, in contrast with the valid time of facts, which are provided by the user. This derives from the different semantics of transaction time and valid time. Specifically, when a fact is (logically) deleted from a table with transaction-time support, its transaction stop time is set automatically by the DBMS to the current time. When a fact is inserted into the table, its transaction start time is set by the DBMS, again to the current time. An update is treated, concerning the transaction-time timestamps, as a deletion followed by an insertion. The transaction time of a database modification must be consistent with the serialization order of the transaction that effects the modification.

Example 3: We can alter the employee table discussed in [3] to be a table with both valid-time and transaction-time support, by adding transaction-time support. □

Temporal upward compatibility guarantees that conventional, nontemporal queries, updates, etc. work as before, with the same semantics.

Since the history of the database is recorded in tables with both valid-time and transaction-time support, we can find out when corrections were made, using a nonsequenced transaction query.

EXAMPLE 4: The query, "When was the street corrected?", combines nonsequenced transaction semantics (since this involves two transaction states: before and after the correction) with sequenced valid semantics. □

EXAMPLE 5: To extract all the information from the employee table, we can use a sequenced valid/sequenced transaction query. Such queries can have arbitrarily complex predicates. "When did we think that someone lived somewhere for more than six months?" □

Modifications take effect at the current transaction time. However, we can still specify the scope of the change in valid time, both before and after now (retroactive and postactive changes, respectively).

EXAMPLE 6: Lilian moved last June 1. □

Finally, arbitrarily complex queries in transaction time can be expressed with nonsequenced transaction queries.

EXAMPLE 7: The query, "When was an employee's address for 1995 corrected?", involves nonsequenced transaction semantics and sequenced valid semantics, with a temporal scope of 1995. □

As always, the concepts also apply to views, cursors, constraints, and assertions.

EXAMPLE 8: The assertion, "An entry in the security table can never be updated. It can only be deleted, and a new entry, with another key value, inserted.", can be expressed with a nonsequenced transaction semantics, stating in effect that the key value is unique over all transaction time. □

6 Supporting Transaction-Time in SQL3

This section informally introduces the new constructs of SQL/Temporal. We build upon the examples given in the previous change proposal [3].

6.1 SQL3 Extensions

We employ an existing reserved word, `TRANSACTION`, whose use parallels that of `VALID`. This reserved word can appear in a number of locations.

Table creation The create table statement is extended to define tables with either or both of valid-time and transaction-time support, through the use of "`AS TRANSACTION`".

Temporal upward compatibility TUC is ensured through the semantics of the language; no new syntax is needed. A transaction-time or table with both valid-time and transaction-time support is transaction timesliced to now to retrieve the data as best known.

Sequenced transaction semantics Sequenced transaction semantics is specified by prepending the reserved word `TRANSACTION`, as with sequenced valid semantics. This applies to queries, views, cursors, assertions, and constraints.

Nonsequenced transaction semantics Nonsequenced transaction semantics is specified by prepending `NONSEQUENCED TRANSACTION`, as with valid time.

Assertion definition A nonsequenced transaction assertion applies simultaneously to all of the states of the underlying table(s). This is in contrast to a snapshot assertion, which is evaluated only on the current state. In both cases, the assertion is checked before a transaction is committed. The fact that tables with transaction-time support are append-only presents an opportunity to optimize the checking of such assertions.

Derived table in a from clause In the from clause, one can also specify `TRANSACTION`. This is the means of converting a table with transaction-time support to and from a table with no temporal support.

Table and column constraints When specified with `NONSEQUENCED TRANSACTION`, such constraints must apply to all states in transaction time, together, of a table with transaction-time support.

Cursor expression Cursors can range over the result of a nonsequenced transaction select. Note however that rows that are not current cannot be updated.

Optional period expression An optional period expression after `TRANSACTION` (without `NONSEQUENCED`) specifies that the transaction-time period of the result is intersected with the value of the expression. This allows one to restrict the result of a select statement, assertion definition, table constraint, column constraint, cursor expression, or view definition to a specified period.

Value expression The value expression "`TRANSACTION(`<correlation name>`)`" evaluates to the transaction-time period of the row associated with the correlation or table name. This is required because transaction-time periods of tables with transaction-time support are not explicit columns (the alternative violates temporal upward compatibility).

Fetch statement The transaction-time period associated with a row with transaction-time support can be placed in a local variable in embedded SQL.

6.2 Overview of the Semantics

The semantics is dictated by three simple rules.

- The absence of `VALID` (respectively, `TRANSACTION`) indicates temporal upward compatibility. The result does not include valid-time (resp. transaction-time) support.
- `VALID` (respectively, `TRANSACTION`) indicates sequenced valid (resp., transaction) semantics. An optional period expression temporally scopes the result. The result includes valid-time (resp., transaction-time) support.
- `NONSEQUENCED` denotes nonsequenced valid (resp., transaction) semantics. An optional period expression after `NONSEQUENCED VALID` provides a valid-time timestamp, yielding valid-time support in the result.

Table 1 summarizes how these options are used to enforce temporal upward compatibility (TUC), sequentiality (SEQ), and non-sequentiality (NONSEQ) respectively. The table also lists the type of the resulting table. Note that we have omitted the equivalent permutations where `TRANSACTION` comes before `VALID`. We abbreviate `VALID` to `VT` , `TRANSACTION` to `TT` , and `NONSEQUENCED` to `NS` . Users need not memorize this table, as each entry is the logical consequence of the above three rules.

The following quick tour provides examples of these constructs.

6.3 A Quick Tour

This quick tour starts with the database as it was when we last left it, at the end of the previous quick tour [3]. The `employee` table has the following contents. Recall that closed-open periods are used here for the valid and transaction periods.

ename	eno	street	city	birthday	Valid
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)

Syntax	Semantics		Result	
	vt	tt	valid	trans
<query expression>	TUC	TUC	—	—
`VT` <query expression>	SEQ	TUC	√	—
`VT` <value exp><query expression>	SEQ	TUC	√	—
`NS VT` <query expression>	NONSEQ	TUC	—	—
`NS VT` <value exp><query expression>	NONSEQ	TUC	√	—
`TT` <query expression>	TUC	SEQ	—	√
`TT` <value exp><query expression>	TUC	SEQ	—	√
`NS TT` <query expression>	TUC	NONSEQ	—	—
`VT AND TT` <query expression>	SEQ	SEQ	√	√
`VT AND TT` <value exp><query expression>	SEQ	SEQ	√	√
`VT` <value exp>`AND TT` <query expression>	SEQ	SEQ	√	√
`VT` <value exp>`AND TT` <value exp><query expression>	SEQ	SEQ	√	√
`VT AND NS TT` <query expression>	SEQ	NONSEQ	√	—
`VT` <value exp>`AND NS TT` <query expression>	SEQ	NONSEQ	√	—
`NS VT AND TT` <query expression>	NONSEQ	SEQ	—	√
`NS VT AND TT` <value exp><query expression>	NONSEQ	SEQ	—	√
`NS VT` <value exp>`AND TT` <query expression>	NONSEQ	SEQ	√	√
`NS VT` <value exp>`AND TT` <value exp><query expression>	NONSEQ	SEQ	√	√
`NS VT AND NS TT` <query expression>	NONSEQ	NONSEQ	—	—
`NS VT` <value exp>`AND NS TT` <query expression>	NONSEQ	NONSEQ	√	—

Table 1: The Various Combinations

The `salary` table has the following contents.

eno	amount	Valid
6542	3200	[1995-02-01 - 1995-06-01)
6542	3360	[1995-06-01 - 1995-07-01)
6542	3360	[1996-01-01 - 9999-12-31)
3463	3400	[1995-02-02 - 1995-04-01)
3463	3570	[1995-04-01 - 9999-12-31)

We can alter the `employee` table to be a table with both valid-time and transaction-time support, by adding transaction-time support. Assume that the current date is July 1, 1995.

```
ALTER TABLE employee ADD TRANSACTION;
COMMIT;
```

Since `employee` was a table with valid-time support, this statement converts it to the following table with both valid-time and transaction-time support. Recall that NULL as the ending delimiter of the transaction-time period simply indicates that the row still logically resides in the table, i.e., has not been logically deleted.

ename	eno	street	city	birthday	Valid	Transaction
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)	[1995-07-01 - NULL)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-07-01 - NULL)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)	[1995-07-01 - NULL)

We retain the `salary` table as a table with valid-time support.

Temporal upward compatibility guarantees that conventional, nontemporal queries, updates, etc. work as before, with the same semantics. We can list those for which (currently, as best known) no one makes a higher salary in a different city.

```
SELECT ename
FROM employee AS e1, salary AS s1
WHERE e1.eno = s1.eno
   AND NOT EXISTS (SELECT ename
                   FROM employee AS e2, salary AS s2
                   WHERE e2.eno = s2.eno AND s2.amount > s1.amount
                      AND e1.city <> e2.city)
```

This takes a timeslice in both valid time and transaction time at now, and returns the result: Lilian.

We can also ask for the valid times that this is true, by simply prepending "`VALID`".

```
VALID SELECT ename
FROM employee AS e1, salary AS s1
WHERE e1.eno = s1.eno
   AND NOT EXISTS (SELECT ename
                   FROM employee AS e2, salary AS s2
                   WHERE e2.eno = s2.eno AND s2.amount > s1.amount
                      AND e1.city <> e2.city)
```

This returns a table with valid-time support, evaluated with sequenced valid semantics, after the current transaction timeslice has been taken.

ename	Valid
Franziska	[1995-02-01 - 1995-02-02)
Lilian	[1995-02-02 - 1995-04-01)
Lilian	[1995-04-01 - 9999-12-31)

There are two rows for Lilian, because two rows of `salary` participated in computing the result. Interestingly, Franziska satisfied the where condition for exactly one day, before Lilian was hired.

Temporally upward compatible modifications also work as before. Assume it is now August 1, 1995. Franziska just moved.

```
UPDATE employee
SET street = 'Niederdorfstrasse 2'
WHERE ename = 'Franziska';
COMMIT;
```

This update yields the following `employee` table.

ename	eno	street	city	birthday	Valid	Transaction
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)	[1995-07-01 - NULL)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-07-01 - 1995-08-01)
Franziska	6542	Niederdorfstrasse 2	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-08-01 - NULL)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)	[1995-07-01 - NULL)

Since the history of the database is recorded in tables with both valid-time and transaction-time support, we can find out when corrections were made, using a nonsequenced transaction query. Assume it is now September 1, 1995.

The query, "When was the street corrected?", combines nonsequenced transaction semantics with sequenced valid semantics.

```
NONSEQUENCED TRANSACTION AND VALID
SELECT e1.ename, e1.street AS old_street, e2.street AS new_street,
        BEGIN(TRANSACTION(e2)) AS trans_time
FROM employee AS e1, employee AS e2
WHERE e1.eno = e2.eno AND TRANSACTION(e1) MEETS TRANSACTION(e2)
```

This yields the following table with valid-time support. The `trans_time` column specifies when the change was made; the implicit timestamp indicates the valid-time period of the fact that was changed.

ename	old_street	new_street	trans_time	Valid
Franziska	Rennweg 683	Niederdorfstrasse 2	1995-08-01	[1996-01-01 - 9999-12-31)

To extract all the information from the `employee` table, we can use a sequenced valid/sequenced transaction query. "When did we think that someone lived somewhere for more than six months?".

```
VALID AND TRANSACTION SELECT ename, street
FROM employee
WHERE INTERVAL(VALID(employee) MONTH) > INTERVAL '6' MONTH
```

ename	street	Valid	Transaction
Franziska	Rennweg 683	[1996-01-01 - 9999-12-31)	[1995-07-01 - 1995-08-01)
Franziska	Niederdorfstrasse 2	[1996-01-01 - 9999-12-31)	[1995-08-01 - 1995-09-01)
Lilian	46 Speedway	[1995-02-02 - 9999-12-31)	[1995-07-01 - 1995-09-01)

Notice that in the result the ending transaction time for data in the current state is always the current time, rather than NULL, reflecting information currently known.

Modifications take effect at the current transaction time. However, we can still specify the scope of the change in valid time, both before and after now (retroactive and postactive changes, respectively).

Assume it is now October 1, 1995. Lilian moved last June 1.

```
VALID PERIOD '[1995-06-01 - 9999-12-31)' UPDATE employee
SET street = '124 Alberca'
WHERE ename = 'Lilian'
COMMIT;
```

This update yields the following `employee` table.

ename	eno	street	city	birthday	Valid	Transaction
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1995-02-01 - 1995-07-01)	[1995-07-01 - NULL)
Franziska	6542	Rennweg 683	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-07-01 - 1995-08-01)
Franziska	6542	Niederdorfstrasse 2	Zurich	1963-07-04	[1996-01-01 - 9999-12-31)	[1995-08-01 - NULL)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 9999-12-31)	[1995-07-01 - 1996-10-01)
Lilian	3463	46 Speedway	Tucson	1970-03-09	[1995-02-02 - 1995-06-01)	[1995-10-01 - NULL)
Lilian	3463	124 Alberca	Tucson	1970-03-09	[1995-06-01 - 9999-12-31)	[1995-10-01 - NULL)

Finally, arbitrarily complex queries in transaction time can be expressed with nonsequenced transaction queries.

The query, "When was an employee's address for 1995 corrected?", involves nonsequenced transaction semantics and sequenced valid semantics, with a temporal scope of 1995. Assume that it is November 1, 1995.

```
NONSEQUENCED TRANSACTION AND VALID PERIOD '[1995-01-01 - 1996-01-01)'
SELECT e1.ename, e1.street AS old_street, e2.street AS new_street,
        BEGIN(TRANSACTION(e2)) AS trans_time
FROM employee AS e1, employee AS e2
WHERE e1.eno = e2.eno AND TRANSACTION(e1) MEETS TRANSACTION(e2)
      AND e1.street <> e2.street
```

This evaluates to the following result, which has an explicit column denoting the date the change was made, and an implicit valid time indicating the time in reality in question.

ename	old_street	new_street	trans_time	Valid
Lilian	46 Speedway	124 Alberca	1995-10-01	[1995-06-01 - 1996-01-01)

Note that the period from February through May is not included in the valid time, as the street didn't change for that period.

As always, the concepts also apply to views, cursors, constraints, and assertions.

The assertion, "An entry in the security table can never be updated. It can only be deleted, and a new entry, with another key value, inserted.", can be expressed with a nonsequenced transaction semantics, stating in effect that the key value is unique over all transaction time.

```
CREATE TABLE security (
        keyvalue NUMERIC(8) NONSEQUENCED TRANSACTION UNIQUE,
        ...
)
```

7 Formal Semantics of SQL/Temporal

We provide a denotational mapping of queries with these language extensions to temporal relational and relational algebra expressions.

We use $\langle t \| VT\rangle$, $\langle t \| TT\rangle$, and $\langle t \| VT, TT\rangle$ to denote a tuple variable ranging over a table with valid-time support, with transaction-time support, and with both valid-time and transaction-time support, respectively. The vertical double-bar "$\|$" is used to separate transaction and valid-time from explicit attributes.

Finally, we use four simple auxiliary functions: $\tau^{vt}, \tau^{tt}, SN^{vt}$, and SN^{tt}. The timeslice operation τ_c computes the timeslice of a table at time c, i.e., it selects all tuples with a timestamp that overlaps chronon c. SN turns an implicit time dimension into an explicit attribute. Both functions are defined for valid and transaction time. Table 2 gives their semantics over all possible table types. Note that SN, which converts an implicit dimension into an explicit attribute, is not needed at the implementation level, where a time dimension is represented simply as an extra column.

	snapshot	valid time	transaction time
$\tau_c^{vt}(r)$	$\{\langle t\rangle \mid \langle t\rangle \in r\}$	$\{\langle t\rangle \mid \langle t \| VT\rangle \in r \wedge VT \textit{ overlaps } c\}$	$\{\langle t \| TT\rangle \mid \langle t \| TT\rangle \in r\}$
$\tau_c^{tt}(r)$	$\{\langle t\rangle \mid \langle t\rangle \in r\}$	$\{\langle t \| VT\rangle \mid \langle t \| VT\rangle \in r\}$	$\{\langle t\rangle \mid \langle t \| TT\rangle \in r \wedge TT \textit{ overlaps } c\}$
$SN^{vt}(r)$	$\{\langle t\rangle \mid \langle t\rangle \in r\}$	$\{\langle t, VT\rangle \mid \langle t \| VT\rangle \in r\}$	$\{\langle t \| TT\rangle \mid \langle t \| TT\rangle \in r\}$
$SN^{tt}(r)$	$\{\langle t\rangle \mid \langle t\rangle \in r\}$	$\{\langle t \| VT\rangle \mid \langle t \| VT\rangle \in r\}$	$\{\langle t, TT\rangle \mid \langle t \| TT\rangle \in r\}$

	valid and transaction time
$\tau_c^{vt}(r)$	$\{\langle t \| VT, TT\rangle \mid \langle t \| VT, TT\rangle \in r \wedge VT \textit{ overlaps } c\}$
$\tau_c^{tt}(r)$	$\{\langle t \| VT, TT\rangle \mid \langle t \| VT, TT\rangle \in r \wedge TT \textit{ overlaps } c\}$
$SN^{vt}(r)$	$\{\langle t, VT \| TT\rangle \mid \langle t \| VT, TT\rangle \in r\}$
$SN^{tt}(r)$	$\{\langle t, TT \| VT\rangle \mid \langle t \| VT, TT\rangle \in r\}$

Table 2: Snapshot Functions and Functions to Convert a Time Dimension into an Explicit Column

Table 3 gives the denotational semantics for all basic statements listed in Table 1. $[\![$<query expression>$]\!]_{\text{standard}}$ evaluates to the standard relational algebra expression which corresponds to <query expression>. $[\![$<query expression>$]\!]_{\text{X}}$, where $X \in \{vt, tt, bi\}$, is equivalent to $[\![$<query expression>$]\!]_{\text{standard}}$ except that every nontemporal relational algebra operator (e.g., $\bowtie, \sigma, \pi$) is replaced by the corresponding temporal relational algebra operator (e.g., $\bowtie^X, \sigma^X, \pi^X$). The semantics of these algebraic operators is a straightforward extension of the semantics given for the valid-time temporal algebra in [3].

Table 3 does not show the semantics of temporal scoping in transaction time, so we provide this semantics here. (We gave the semantics for temporal scoping in valid time in the previous change proposal [3].)

$$[\![\texttt{TRANSACTION}\ p\ \text{<query expression>}]\!]_{\text{SQL/T}}(r_1, \ldots, r_n) \triangleq$$

$$\{\langle t \mid\mid TT\rangle \mid \langle t \mid\mid TT'\rangle \in [\![\texttt{TRANSACTION}\ \text{<query expression>}]\!]_{\text{SQL/T}}(r_1, \ldots, r_n) \wedge TT = TT' \cap [\![p]\!] \wedge TT \neq \emptyset\}$$

Example 9: The first example is a nontemporal query, i.e., a query evaluated with standard semantics.

$$
\begin{aligned}
&\llbracket \text{<query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{standard}}(\tau^{tt}_{now}(\tau^{vt}_{now}(r_1)), \ldots, \tau^{tt}_{now}(\tau^{vt}_{now}(r_n)))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{VT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{vt}}(\tau^{tt}_{now}(r_1), \ldots, \tau^{tt}_{now}(r_n))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{NS VT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{standard}}(\tau^{tt}_{now}(SN^{vt}(r_1)), \ldots, \tau^{tt}_{now}(SN^{vt}(r_n)))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{TT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{tt}}(\tau^{vt}_{now}(r_1), \ldots, \tau^{vt}_{now}(r_n)
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{NS TT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{standard}}(\tau^{vt}_{now}(SN^{tt}(r_1)), \ldots, \tau^{vt}_{now}(SN^{tt}(r_n)))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{VT AND TT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{bi}}(r_1, \ldots, r_n)
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{VT AND NS TT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{vt}}(SN^{tt}(r_1), \ldots, SN^{tt}(r_n))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{NS VT AND TT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{tt}}(SN^{vt}(r_1)), \ldots, SN^{vt}(r_n))
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \texttt{NS VT AND NS TT} \text{ <query expression>} \rrbracket_{\mathrm{SQL/T}}(r_1, \ldots, r_n) \triangleq \\
&\qquad \llbracket \text{<query expression>} \rrbracket_{\mathrm{standard}}(SN^{tt}(SN^{vt}(r_1), \ldots, SN^{tt}(SN^{vt}(r_n)))
\end{aligned}
$$

Table 3: Denotational Semantics

Assume that p and q are tables with both valid-time and transaction-time support. The query Q_1

```
NONSEQUENCED VALID
  SELECT p.X
  FROM p, q
  WHERE p.X = q.X
```

is equivalent to the relational algebra expression

$$[\![Q_1]\!]_{\text{SQL/TEMPORAL}}(p, q) = \pi_{p.X}\sigma_{p.X=q.X}(SN^{vt}(\tau^{tt}_{now}(p) \times SN^{vt}(\tau^{tt}_{now}(q)))) \ .$$

□

EXAMPLE 10: The second example is the query Q_2 evaluated with temporal semantics.

```
VALID AND TRANSACTION
  SELECT p.X
  FROM p, q
  WHERE p.X = q.X
```

is equivalent to the temporal relational algebra expression

$$[\![Q_2]\!]_{\text{SQL/TEMPORAL}}(p, q) = \pi^{bi}_{p.X}(p \bowtie^{bi}_{p.X=q.X} q) \ .$$

Note that apart from the superscripts, which are added to relational algebra operators, the translation between SQL queries and relational algebra expressions has not changed at all. □

8 Implementing These Extensions

Unlike valid time, transaction time cannot be entirely simulated with tables with explicit timestamp columns. The reason is that tables with transaction-time support are *append-only*: they grow monotonically. While the query functionality can be simulated on table with no temporal support, in the same way that valid-time query functionality can be translated into queries on table with no temporal support, there is no way to restrict the user to modifications that ensure the table is append-only. While one can revoke permission to use DELETE, it is still possible for the user to corrupt the transaction timestamp via database updates and insertions. This means that the user can never be sure that what the table says was stored at some time in the past was actually in the table at that time. The only way to ensure the consistency of the data is to have the DBMS maintain the transaction timestamps automatically.

The semantics is straightforward. An insertion has a transaction timestamp of

```
PERIOD '[CURRENT_TIMESTAMP - NULL]'
```

When the row is extracted from the table, the terminating value can be replaced with the (then) CURRENT_TIMESTAMP via a COALESCE operation.

A deletion replaces the ending time of the transaction timestamp with CURRENT_TIMESTAMP. An update has the semantics of a deletion followed by an insertion.

9 Summary

This change proposal builds on the previous change proposal [3], introducing transaction time as well as tables with transaction-time support, sequenced transaction semantics, nonsequenced transaction semantics, scoping on transaction time via an optional period expression, and modification semantics. The specific syntactic additions were outlined and examples given to illustrate these constructs. We provided a formal semantics, in terms of the formal semantics of SQL3, for the new constructs.

We end by listing some of the advantages of the approach espoused here.

- No new reserved words are required to support transaction time.
- The extensions are compatible with, and orthogonal to, those for valid time.
- A simple period expression permits the transaction-time scope to be specified.
- Nonsequenced transaction semantics permits tables with transaction-time support to be converted to tables with no temporal support with an explicit timestamp column, even within a query.
- A public-domain prototype [4] demonstrates the practical viability of the proposed constructs. The quick tour was validated on this prototype.

10 Proposed Language Extensions

The syntax is given as extensions to "Database Language SQL — Part 7: Temporal," [2] as well as the previous change proposal [3].

11 Clause 3 Definitions, notations, and conventions

11.1 Subclause 3.1 Definitions

1) Add the following terms.

q) **transaction time**: The transaction time of a fact is the time during which the fact was asserted.

Note to proposal reader: It follows that a proposition P, together with an associated valid time TV and an associated transaction time TT, is equivalent to the proposition " 'P is true during TV' was asserted during TT".

r) **transaction-time precision**: This is an implementation-defined precision.

s) **transaction-time row**: A row of a table with an associated transaction time, which is a value of data type PERIOD, of the transaction-time precision.

t) **table has transaction-time support**: A table has transaction-time support if each row is a transaction-time row.

u) **state of a table with transaction-time support at a transaction time**: The state of a table with transaction-time support, TT, at a specified time, T, is the table without transaction-time support comprising the rows of TT associated with transaction-time periods that overlap T.

v) **current state of a table with transaction-time support**: The current state of a table with transaction-time support is the state of that table at transaction time CURRENT_TIMESTAMP.

12 Clause 4 Concepts

1) Insert this new Subclause immediately after Subclause 4.4, "Concepts Used in Modification Statements".

12.1 Subclause 4.5 Semantics of Statements on Tables with Transaction-Time Support

The semantics of the statements on tables with transaction-time support is defined in terms of the semantics of statements on tables without transaction-time support, which is already defined in this International Standard. In this way, the definitions exploit the other definitions and rules in this International Standard, without replication.

In this Clause the meaning is described only for queries, but the concepts apply to views, cursors, constraints and assertions.

Transaction-time upward compatible queries (i.e., SELECT without TRANSACTION) treat each underlying table that has transaction-time support as a table without transaction-time support, by using instead the current state of the table. Hence, a query evaluated with temporal upward compatibility on a table with transaction-time support will use only the current state in the evaluation.

Sequenced transaction semantics (e.g., TRANSACTION SELECT) is used only on tables with transaction-time support, and queries with sequenced transaction semantics result in tables with transaction-time support. Sequenced transaction semantics is defined in terms of the semantics of the statement on tables without transaction-time support. Let Q be a sequenced transaction query, with Q = TRANSACTION Q1, where Q1 is a query without TRANSACTION. The meaning of Q1 on tables without temporal support is already defined by this International Standard. Let R be the table with transaction-time support that is the result of Q on one or more tables with transaction-time support. For all times T, the state of R at time T is the result of evaluating Q1 on the states of the underlying tables at time T. Any R that satisfies this property is a transaction result of Q. Note that this discussion holds regardless of whether Q involves valid time, i.e., whether Q effects valid-time upward compatibility, sequenced valid semantics, or nonsequenced valid semantics.

Nonsequenced transaction semantics (e.g., NONSEQUENCED TRANSACTION SELECT) treats each underlying table that has transaction-time support as a table without transaction-time support, but with an additional unnamed column whose value for a row in the table is the transaction-time period associated with the corresponding row in the original table.

The <value expression> following TRANSACTION specifies the "temporal scope" of the sequenced transaction query or integrity constraint. For queries, the result is computed with sequenced transaction semantics, then the transaction-time periods of the result are intersected with the value of the <value expression> to determine the final transaction-time period. For integrity constraints, the effect of the constraint or modification is restricted to the value of <value expression>.

Modifications on tables with transaction-time support are always performed on the current state of the table, with the resulting rows of the new state having a transaction-time period with a beginning delimiting timestamp of CURRENT_TIMESTAMP, thereby ensuring the append-only nature of transaction-time support. For updates and deletions, the terminating delimiting timestamps of the rows that are affected are set to the granule preceding CURRENT_TIMESTAMP.

Note to proposal reader: This semantics is orthogonal to and consistent with the semantics presented in Subclauses 4.3 and 4.4 for valid time.

13 Clause 6 Scalar expressions

13.1 Subclause 6.1 <item reference>

Function

Reference a column, parameter, or variable.

Format

No additional Format items.

Syntax Rules

1. (Insert this SR) If a <column name> CN is contained in a <transaction expression> simply contained in a <query expression> QE, then the scope clause SC of CN is the <query expression body> simply contained in QE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

14 Section 6.4 <period value function>

Function

Specify a function yielding a value of type period.

Format

1) In the Format, replace the <period value function> BNF production with:

<period value function> ::=
!! All alternatives from Parts 2 and 7 of this International Standard
| <transaction function>

2) Add the following BNF production:

<transaction function> ::=
| TRANSACTION <left paren> { <table name> | <correlation name> } <right paren>

Syntax Rules

1. (Insert this SR) The <table name> or <correlation name> of a <transaction function> shall be associated with a table with transaction-time support.
2. (Insert this SR) The data type of <transaction function> shall be PERIOD.

Note to proposal reader: The precision of the transaction-time period timestamp of a table with transaction-time support was defined in Subclause 3.1 "Definitions".

Access Rules

No additional Access Rules.

General Rules

1. (Insert this SR) The <transaction function> evaluates to the transaction time of the relevant row of the table associated with the <table name> or <correlation name>.

15 Clause 7 Query Processing

15.1 Subclause 7.14 <query expression>

Function

Specify a table.

Format

1) In the Format, replace the <query expression> BNF production with:

```
<time option> ::=
        <valid option> [ AND <transaction option> ]
      |     <transaction option> [ AND <valid option> ]
```

Note to proposal reader: This adds an optional <transaction option> either before or following the <valid option> to <time option>.

2) Add the following BNF production:

```
<transaction option> ::=
        [ NONSEQUENCED ] TRANSACTION [ <value expression> ]
```

Note to proposal reader: This syntax is symmetric with that for <valid option>.

Syntax Rules

1. (Insert this SR) The data type of <value expression> of <transaction option> shall be a period data type.
2. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified in <query expression>, then each exposed table, query, or correlation name contained in the <query expression body> without an intervening <from clause> shall identify a table with transaction-time support.

   Note to proposal reader: This ensures that sequenced transaction queries are only evaluated "over" tables with transaction-time support.

3. (Insert this SR) If TRANSACTION is specified in the <time option> of a <query expression> Q, then either Q shall be simply contained in a <from clause> or Q shall not be contained in a second query expression.

   Note to proposal reader: TRANSACTION is allowed in the same places that VALID is permitted.

4. (Insert this SR) If NONSEQUENCED is specified in a <transaction option> TO, then TO shall not contain a <value expression>.
5. (Insert this SR) The data type of the <value expression> contained in <time option> shall be PERIOD, with a precision of the transaction-time precision. Subclause 6.3 "<item reference>" restricts the scope of column names in <value expression>.
6. (Insert this SR) Let T be the result of the <query expression>.

   Case:

   a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then T shall be a table with transaction-time support.

b) If NONSEQUENCED TRANSACTION is specified in <time option>, then T shall be a table without transaction-time support.

c) Otherwise, T shall be a table without transaction-time support.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:

    a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then <query expression body> is evaluated with sequenced transaction semantics. If <value expression> is specified in the <transaction option> of <time option>, then for each resulting row, the transaction-time period of the row is the intersection of the value of <value expression> and the original transaction-time period of the row.

    b) If NONSEQUENCED TRANSACTION is specified in <time option>, then <query expression body> is evaluated with nonsequenced transaction semantics.

    c) Otherwise, the <query expression body> is evaluated with transaction-time upward compatibility.

16 Clause 9 Schema definition and manipulation

16.1 Subclause 9.2 <table definition>

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

Format

1) In the Format, replace the <temporal definition> BNF production with:

```
<temporal definition> ::=
        AS VALID [ <table precision> ] [ AND TRANSACTION ]
      |     AS TRANSACTION [ AND VALID [ <table precision> ] ]
```

Note to proposal reader: This augments the production for the non-terminal <temporal definition> with an additional, optional clause to specify that the new table is to be a table with transaction-time support. No <table precision> for transaction time can be specified, as that is supplied by the implementation.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:

   a) If TRANSACTION is specified in <temporal definition>, then the table is a table with transaction-time support.

   b) Otherwise, the table does not have transaction-time support.

16.2 Subclause 9.3 <column definition>

Function

Define a column of a table.

Format

No additional Format items.

Note to proposal reader: TRANSACTION is now allowed in <time option>.

Syntax Rules

1. (Insert this SR) If TRANSACTION is specified in <time option>, then T shall be a table with transaction-time support.
2. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified,

   Case:

   a) If <column constraint> is <references specification>, then the table identified by <table name> simply contained in the <referenced table and columns> of <references specification> shall be a table with transaction-time support.

   b) If <column constraint> is <check constraint definition>, then each table associated with an exposed <table name>, <query expression>, or <correlation name> contained in the <column constraint> without an intervening <from clause> shall be a table with transaction-time support and with identical precision.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:

   a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then <column constraint> is evaluated with sequenced transaction semantics. If <value expression> is contained in the <transaction option> of <time option>, then the <column constraint> of a <column constraint definition> applies only to those states at transaction times overlapping the value of <value expression>.

   b) If NONSEQUENCED is specified in <time option>, then <column constraint> is evaluated with nonsequenced transaction semantics.

   c) Otherwise, <column constraint> is evaluated with transaction-time upward compatibility.

16.3 Subclause 9.4 <table constraint definition>

Function

Specify an integrity constraint.

Format

No additional Format items.

Note to proposal reader: TRANSACTION is now allowed in <time option>.
For constraints and assertions, there are four cases:

1. `CHECK`
   - works on anything
   - only considers current state
2. `TRANSACTION CHECK`
   - works only on tables with transaction-time support
   - the assertion must be true for the state at every transaction time
3. `TRANSACTION` <period exp> `CHECK`
   - like `TRANSACTION CHECK`, but only considers the times in <period exp> (a simple example is `TRANSACTION PERIOD '[1995-01-01 - 1995-12-31]' CHECK`)
4. `NONSEQUENCED TRANSACTION CHECK`
   - works on anything
   - acts like tables with transaction-time support have an explicit (unnamed) timestamp column; all rows are considered at once

`NONSEQUENCED TRANSACTION` <period exp> `CHECK` is not allowed.
End of note.

Syntax Rules

1. (Insert this SR) If TRANSACTION is specified in <time option>, then T shall be a table with transaction-time support.
2. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified in <table constraint definition>, then each exposed table, query, or correlation name contained in the <table constraint> without an intervening <from clause> shall identify a table with transaction-time support and with identical transaction-time period precision.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:

a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then <table constraint> is evaluated with sequenced transaction semantics. If <value expression> is contained in the <transaction option> of <time option>, then the <table constraint> of a <column constraint definition> only applies to those states at transaction times overlapping the value of <value expression>.

b) If NONSEQUENCED TRANSACTION is specified in <time option>, then <table constraint> is evaluated with nonsequenced transaction semantics.

c) Otherwise, <table constraint> is evaluated with transaction-time upward compatibility.

16.4 Subclause 9.5 <alter table statement>

Function

Change the definition of a table.

Format

1) In the Format, add the following BNF production:

```
<alter table action> ::=
          !! All alternatives from ISO/EIC 9075 and from part 7
        | <add transaction definition>
        | <drop transaction definition>
```

2) Add the following productions:

```
<add transaction definition> ::=
          ADD TRANSACTION

<drop transaction definition> ::=
          DROP TRANSACTION
```

Syntax Rules

1. (Insert this SR) For the <add transaction definition>, *T* shall not have transaction-time support.
2. (Insert this SR) For the <drop transaction definition>, *T* shall be a table with transaction-time support.

Access Rules

No additional Access Rules.

General Rules

1. (Add this GR) If <add transaction definition> is specified, then transaction-time support is added to each row of T, by associating with that row a transaction-time period from the current timestamp to forever with a precision of transaction-time precision.
2. (Insert this GR) If <drop transaction definition> is specified, then

   Case:

   a) If T has valid-time support, then transaction-time support is removed from T by replacing T with the result of
      VALID SELECT * FROM T
   b) Otherwise, transaction-time support is removed from T by replacing T with the result of
      SELECT * FROM T

   Note to proposal reader: That is, only the current state is retained.

16.5 Subclause 9.6 <assertion definition>

Function

Specify an integrity constraint by means of an assertion and specify when the assertion is to be checked.

Format

No additional Format items.

Note to proposal reader: TRANSACTION is now allowed in <time option>.

Syntax Rules

1. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified in <time option>, then each exposed table, query, or correlation name contained in the <search condition> without an intervening <from clause> shall identify a table with transaction-time support.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:

   a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then <search condition> is evaluated with sequenced transaction semantics. If <value expression> is contained in the <transaction option> of <time option>, then the <triggered assertion> of a <column constraint definition> only applies to those states at times overlapping the value of <value expression>

   b) If NONSEQUENCED TRANSACTION is specified in <time option>, then <search condition> is evaluated with nonsequenced transaction semantics.

   c) Otherwise, <search condition> is evaluated with transaction-time upward compatibility.

17 Clause 11 Data manipulation

17.1 Subclause 11.1 <fetch statement>

Function

Position a cursor on a specified row of a table and retrieve values from that row.

Format

1) In the Format, replace the <fetch statement> BNF production with:

```
<fetch statement> ::=
        FETCH [ [ <fetch orientation> ] FROM ] <cursor name>
        [ INTO <fetch target list> ]
             [ <fetch stamp>]
```

2) Insert the following BNF production:

```
<fetch stamp> ::=
        INTO VALID <target specification> [ INTO TRANSACTION <target specification> ]
        |    INTO TRANSACTION <target specification> [ INTO VALID <target specification> ]
```

Note to proposal reader: This extends the <fetch statement> to also allow the transaction-time period of the row to be accessed.

Syntax Rules

1. (Replace SR 1 with this SR) <fetch statement> shall contain "INTO <fetch target list>", "INTO VALID <target specification>", "INTO TRANSACTION <target specification>", or any combination of these.
2. (Insert this SR) If INTO TRANSACTION is specified in the <fetch statement>, then T shall have transaction-time support. The data type of the <target specification> of <fetch statement> shall be PERIOD with a precision of the transaction-time precision.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) If INTO TRANSACTION is specified, then the valid-time period associated with the current row is assigned to the target of the <target specification>.

17.2 Subclause 11.2 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

No additional Format items.

Note to proposal reader: TRANSACTION is now allowed in <time option>.

Syntax Rules

1. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified in <time option>, then each exposed <table name>, <query expression>, or <correlation name> contained in the <table expression> without an intervening <from clause> shall identify a table with transaction-time support.
2. (Insert this SR) If TRANSACTION is specified in a <time option> of a <query expression> Q that is contained in the <table expression> of <select statement: single row>, then Q shall be simply contained in a <from clause>.
3. (Insert this SR) Let T be the result of <select statement: single row>.

   Case:

   a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then T shall be a table with transaction-time support and with precision P. The precision of <value expression> of the <transaction option> of <time option> shall be P.
   b) If NONSEQUENCED TRANSACTION is specified in <time option>, then Case:
      i) If <value expression> is specified in the <transaction option> of <time option>, then T shall be a table with transaction-time support and with a precision of <value expression>.
      ii) Otherwise, T shall be a table without transaction-time support.
   c) Otherwise, T shall be a table without transaction-time support.
4. (Insert this SR) The scope for the <transaction expression> contained in the <time option> of <select statement: single row> shall be the columns of the <select list>.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:

   a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then the <query expression body> is evaluated with sequenced transaction semantics. If <value expression> is specified in the <transaction option> of <time option>, then for each resulting row, the transaction-time period of the row is the intersection of the transaction of <value expression> and the original transaction-time period of the row.
   b) If NONSEQUENCED TRANSACTION is specified in <time option>, then the <query expression body> is evaluated with nonsequenced transaction semantics.
   c) Otherwise, the <query expression body> is evaluated with transaction-time upward compatibility.

17.3 Subclause 11.3 <delete statement: positioned>

Function

Delete a row of a table.

Format

No additional Format items.

Syntax Rules

1. (Insert this SR) TRANSACTION shall not be specified in <time option>.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) If T is a table with transaction-time support, then to effectively delete a row, the ending delimiting timestamp of the row is set to the granule preceding CURRENT_TIMESTAMP.

17.4 Subclause 11.4 <delete statement: searched>

Function

Delete rows of a table.

Format

No additional Format items.

Note to proposal reader: TRANSACTION is now allowed in <time option>.

Syntax Rules

1. (Insert this SR) If TRANSACTION is specified in <time option>, then T shall be a table with transaction-time support.
2. (Insert this SR) If TRANSACTION is specified in a <time option> of a <query expression> Q that is contained in the <search condition> of <delete statement: searched>, then Q shall be simply contained in a <from clause>.
3. (Insert this SR) A <value expression> shall not be contained in the <transaction option> of <time option>.
4. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified in <time option>, then each exposed <table name>, <query expression>, or <correlation name> contained in the <search condition> without an intervening <from clause> shall identify a table with transaction-time support.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:
   a) If TRANSACTION without NONSEQUENCED is specified in <time option>, then the <search condition> is evaluated with transaction-time upward compatibility.
   b) If NONSEQUENCED TRANSACTION is specified in <time option>, then the <search condition> is evaluated with nonsequenced transaction semantics.
   c) Otherwise, the <search condition> is evaluated with transaction-time upward compatibility.
2. (Insert this GR) If T is a table with transaction-time support, then to effectively delete a row, the ending delimiting timestamp of the row is set to the granule preceding CURRENT_TIMESTAMP.

17.5 Subclause 11.5 <insert statement>

Function

Create new rows in a table.

Format

No additional Format items.

Syntax Rules

1. (Insert this SR) <insert columns and source> shall evaluate to a table without transaction-time support.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Each row of the result of <insert columns and source> shall be associated with a transaction-time period from CURRENT_TIMESTAMP to NULL.

17.6 Subclause 11.6 <update statement: positioned>

Function

Update a row of a table.

Format

No additional Format items.

Syntax Rules

1. (Insert this SR) TRANSACTION shall not be specified in <time option>.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) If T is a table with transaction-time support, the ending delimiting timestamp of the current row is set to the granule preceding CURRENT_TIMESTAMP, and a row with the new values for the columns is inserted into T, with an associated transaction timestamp of CURRENT_TIMESTAMP to NULL.

17.7 Subclause 11.7 <update statement: searched>

Function

Update rows of a table.

Format

No additional Format items.

Note to proposal reader: TRANSACTION is now allowed in <time option>.

Syntax Rules

1. (Insert this SR) If TRANSACTION is specified in <time option>, then T shall be a table with transaction-time support.
2. (Insert this SR) If TRANSACTION is specified in a <time option> of a <query expression> Q that is contained in the <search condition> of <update statement: searched>, then Q shall be simply contained in a <from clause>.
3. (Insert this SR) A <value expression> shall not be contained in the <transaction option> of <time option>.
4. (Insert this SR) If TRANSACTION without NONSEQUENCED is specified in <time option>, then each exposed <table name>, <query expression>, or <correlation name> contained in the <search condition> without an intervening <from clause> shall identify a table with transaction-time support.

Access Rules

No additional Access Rules.

General Rules

1. (Insert this GR) Case:
   a) If TRANSACTION is specified without NONSEQUENCED in <time option>, then the <search condition> is evaluated with transaction-time upward compatibility.
   b) If NONSEQUENCED TRANSACTION is specified in <time option>, then the <search condition> is evaluated with nonsequenced transaction semantics.
   c) Otherwise, the <search condition> is evaluated with transaction-time upward compatibility.
2. (Insert this GR) If T is a table with transaction-time support, the ending delimiting timestamp of the current row is set to the granule preceding CURRENT_TIMESTAMP, and a row with the new values for the columns is inserted into T, with an associated transaction timestamp of CURRENT_TIMESTAMP to NULL.

18 Clause 12 Information Schema and Definition Schema

18.1 Subclause 12 Information Schema

18.1.1 Subclause 12.1.1 TABLES view

Function

Identify the tables defined in this catalog that are accessible to a given user.

Definition

1) Replace the TABLES view with the following.

```
CREATE VIEW TABLES
         AS SELECT
               TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE,
               VALID_TIME_SUPPORT, VALID_TIME_PRECISION, TRANSACTION_TIME_SUPPORT,
         FROM DEFINITION_SCHEMA.TABLES
         WHERE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
         IN (
               SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
               FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
               WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
              UNION
               SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
               FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
               WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
               AND TABLE_CATALOG
                = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Note to proposal reader: This adds one column: TRANSACTION_TIME_SUPPORT.

Leveling Rules

No additional Leveling Rules.

18.2 Subclause 12.2 Definition Schema

18.2.1 Subclause 12.2.2 TABLES base table

Function

The TABLES table contains one row for each table including views. It effectively contains a representation of the table descriptors.

Definition

1) Replace the TABLES table with the following.

```
CREATE TABLE TABLES
         (
         TABLE_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TABLE_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TABLE_NAME   INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TABLE_TYPE   INFORMATION_SCHEMA.CHARACTER_DATA,
               CONSTRAINT TABLE_TYPE_NOT_NULL NOT NULL,
         VALID_TIME_SUPPORT CHARACTER_DATA
               CONSTRAINT VALID_TIME_SUPPORT_CHECK
         CHECK (VALID_TIME_SUPPORT IN ('STATE','NONE')),
         VALID_TIME_PRECISION  INFORMATION_SCHEMA.CARDINAL_NUMBER,
         TRANSACTION_TIME_SUPPORT CHARACTER_DATA
               CONSTRAINT TRANSACTION_TIME_SUPPORT_CHECK
         CHECK (TRANSACTION_TIME_SUPPORT IN ('STATE','NONE')),
               CONSTRAINT TABLE_TYPE_CHECK CHECK ( TABLE_TYPE IN
              ( 'BASE TABLE', 'VIEW', 'GLOBAL TEMPORARY',
                'LOCAL TEMPORARY', 'EXTENT' ) ),
        --------------------------------------------------------------
         ISO Only-caused by ANSI changes not yet considered by ISO
         --------------------------------------------------------------
               CONSTRAINT TABLE_TYPE_CHECK CHECK ( TABLE_TYPE IN
              ( 'BASE TABLE', 'VIEW', 'GLOBAL TEMPORARY', 'LOCAL TEMPORARY' ) ),

         --------------------------------------------------------------

               CONSTRAINT CHECK_TABLE_IN_COLUMNS
         CHECK ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
               ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
         FROM COLUMNS ) ),

         CONSTRAINT TABLES_PRIMARY_KEY
               PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

         CONSTRAINT TABLES_FOREIGN_KEY_SCHEMATA
               FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA ) REFERENCES SCHEMATA,

         CONSTRAINT TABLES_CHECK_NOT_VIEW CHECK ( NOT EXISTS
               ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
         FROM TABLES
         WHERE TABLE_TYPE = 'VIEW'
         EXCEPT
         SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
```

```
FROM VIEWS ) )
)
```

Note to proposal reader: This adds one column: TRANSACTION_TIME_SUPPORT.

Leveling Rules

No additional Leveling Rules.

18.2.2 Subclause 12.2.3 VIEWS base table

Function

The VIEWS table contains one row for each row in the TABLES table with a TABLE_TYPE of 'VIEW'. Each row describes the query expression that defines a view. The table effectively contains a representation of the view descriptors.

Definition

1) Replace the VIEWS table with the following.

```
CREATE TABLE VIEWS
         (
         TABLE_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TABLE_SCHEMA INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TABLE_NAME     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         VIEW_DEFINITION INFORMATION_SCHEMA.CHARACTER_DATA,
         CHECK_OPTION INFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT CHECK_OPTION_NOT_NULL NOT NULL
               CONSTRAINT CHECK_OPTION_CHECK CHECK ( CHECK_OPTION IN
         ( 'CASCADED', 'LOCAL', 'NONE' ) ),
         IS_UPDATABLE    INFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT IS_UPDATABLE_NOT_NULL NOT NULL
               CONSTRAINT IS_UPDATABLE_CHECK CHECK ( IS_UPDATABLE IN ( 'YES', 'NO' ) ),
         VALID_TIME_SUPPORT CHARACTER_DATA
       CONSTRAINT VALID_TIME_SUPPORT_CHECK CHECK (VALID_TIME_SUPPORT IN ('STATE','NONE'))
         VALID_TIME_PRECISION  INFORMATION_SCHEMA.CARDINAL_NUMBER,
         TRANSACTION_TIME_SUPPORT CHARACTER_DATA
       CONSTRAINT TRANSACTION_TIME_SUPPORT_CHECK CHECK (TRANSACTION_TIME_SUPPORT IN ('STATE','NONE'))
         CONSTRAINT VIEWS_PRIMARY_KEY
               PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

         CONSTRAINT VIEWS_IN_TABLES_CHECK
               CHECK ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
               ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
         FROM TABLES
         WHERE TABLE_TYPE = 'VIEW' ) ),

         CONSTRAINT VIEWS_IS_UPDATABLE_CHECK_OPTION_CHECK
               CHECK ( ( IS_UPDATABLE, CHECK_OPTION ) NOT IN
              ( VALUES ( 'NO', 'CASCADED' ), ( 'NO', 'LOCAL' ) ) )
         )
```

Note to proposal reader: This adds one column: TRANSACTION_TIME_SUPPORT.

Leveling Rules

No additional Leveling Rules.

18.2.3 Subclause 12.2.4 TABLE_CONSTRAINTS base table

Function

The TABLE_CONSTRAINTS table has one row for each table constraint associated with a table. It effectively contains a representation of the table constraint descriptors.

Definition

1) Replace the TABLE_CONSTRAINTS table with the following.

```
CREATE TABLE TABLE_CONSTRAINTS
         (
         CONSTRAINT_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_NAME         INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_TYPE         INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT CONSTRAINT_TYPE_NOT_NULL NOT NULL
               CONSTRAINT CONSTRAINT_TYPE_CHECK

         CHECK ( CONSTRAINT_TYPE IN
                ( 'UNIQUE',
                  'PRIMARY KEY',
                  'FOREIGN KEY',
                  'CHECK' ) ),

         TABLE_CATALOGINFORMATION_SCHEMA.SQL_IDENTIFIER
               CONSTRAINT TABLE_CONSTRAINTS_TABLE_CATALOG_NOT_NULL NOT NULL,
         TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER
               CONSTRAINT TABLE_CONSTRAINTS_TABLE_SCHEMA_NOT_NULL NOT NULL,
         TABLE_NAME  INFORMATION_SCHEMA.SQL_IDENTIFIER
               CONSTRAINT TABLE_CONSTRAINTS_TABLE_NAME_NOT_NULL NOT NULL,
         IS_DEFERRABLE INFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT TABLE_CONSTRAINTS_IS_DEFERRABLE_NOT_NULL NOT NULL,
         INITIALLY_DEFERRED   INFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT TABLE_CONSTRAINTS_INITIALLY_DEFERRED_NOT_NULL
         VALID_TIME_SUPPORT CHARACTER_DATA
               CONSTRAINT VALID_TIME_SUPPORT_CHECK
         CHECK (VALID_TIME_SUPPORT IN ('SEQUENCED','NONSEQUENCED','NONE')),
         VALID_TIME_PERIOD  INFORMATION_SCHEMA.CARDINAL_NUMBER,
         TRANSACTION_TIME_SUPPORT CHARACTER_DATA
               CONSTRAINT TRANSACTION_TIME_SUPPORT_CHECK
         CHECK (TRANSACTION_TIME_SUPPORT IN ('SEQUENCED','NONSEQUENCED','NONE')),
         TRANSACTION_TIME_PERIOD  INFORMATION_SCHEMA.CARDINAL_NUMBER,

         CONSTRAINT TABLE_CONSTRAINTS_PRIMARY_KEY
               PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

         CONSTRAINT TABLE_CONSTRAINTS_DEFERRED_CHECK
               CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
              ( VALUES ( 'NO',  'NO' ),
              ( 'YES', 'NO' ),
              ( 'YES', 'YES' ) ) ),

         CONSTRAINT TABLE_CONSTRAINTS_CHECK_VIEWS
```

```
         CHECK ( TABLE_CATALOG
        <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
    OR
      ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
 FROM TABLES
 WHERE TABLE_TYPE <> 'VIEW' ) ) ),

   CONSTRAINT TABLE_CONSTRAINTS_UNIQUE_CHECK
         CHECK ( 1 =( SELECT COUNT (*)
FROM ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA,
             CONSTRAINT_NAME FROM TABLE_CONSTRAINTS
WHERE CONSTRAINT_TYPE IN
          ( 'UNIQUE', 'PRIMARY KEY' )
UNION ALL
SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
FROM REFERENTIAL_CONSTRAINTS
UNION ALL
SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
FROM CHECK_CONSTRAINTS ) AS X
WHERE ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
        = ( X.CONSTRAINT_CATALOG, X.CONSTRAINT_SCHEMA, X.CONSTRAINT_NAME ) ) ),

   CONSTRAINT UNIQUE_TABLE_PRIMARY_KEY_CHECK
         CHECK ( UNIQUE ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
        FROM TABLE_CONSTRAINTS
        WHERE CONSTRAINT_TYPE = 'PRIMARY KEY' ) )
   )
```

Note to proposal reader: This adds two columns: TRANSACTION_TIME_SUPPORT and TRANSACTION_TIME_PERIOD.

Leveling Rules

No additional Leveling Rules.

18.2.4 Subclause 12.2.5 CHECK_CONSTRAINTS base table

Function

The CHECK_CONSTRAINTS table has one row for each domain constraint, table check constraint, and assertion.

Definition

1) Replace the CHECK_CONSTRAINTS table with the following.

```
CREATE TABLE CHECK_CONSTRAINTS
         (
         CONSTRAINT_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CHECK_CLAUSE    INFORMATION_SCHEMA.CHARACTER_DATA,
         VALID_TIME_SUPPORT CHARACTER_DATA
         CONSTRAINT VALID_TIME_SUPPORT_CHECK
                CHECK (VALID_TIME_SUPPORT IN ('SEQUENCED','NONSEQUENCED,'NONE'))
         VALID_TIME_PERIOD  INFORMATION_SCHEMA.CARDINAL_NUMBER,
         TRANSACTION_TIME_SUPPORT CHARACTER_DATA
         CONSTRAINT TRANSACTION_TIME_SUPPORT_CHECK
                CHECK (TRANSACTION_TIME_SUPPORT IN ('SEQUENCED','NONSEQUENCED,'NONE'))
         TRANSACTION_TIME_PERIOD  INFORMATION_SCHEMA.CARDINAL_NUMBER,

         CONSTRAINT CHECK_CONSTRAINTS_PRIMARY_KEY
               PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

         CONSTRAINT CHECK_CONSTRAINTS_SOURCE_CHECK
               CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
     IN
       ( SELECT * FROM (
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
               FROM ASSERTIONS
               UNION
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
               FROM TABLE_CONSTRAINTS
               UNION
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
               FROM DOMAIN_CONSTRAINTS ) ) )
         )
```

Note to proposal reader: This adds two columns: TRANSACTION_TIME_SUPPORT and TRANSACTION_TIME_PERIOD.

Leveling Rules

No additional Leveling Rules.

18.2.5 Subclause 12.2.6 ASSERTIONS base table

Function

The ASSERTIONS table has one row for each assertion. It effectively contains a representation of the assertion descriptors.

Definition

1) Replace the TABLE_ASSERTIONS table with the following.

```
CREATE TABLE ASSERTIONS
         (
         CONSTRAINT_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT_NAME         INFORMATION_SCHEMA.SQL_IDENTIFIER,
         IS_DEFERRABLEINFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT ASSERTIONS_IS_DEFERRABLE_NOT_NULL NOT NULL
         INITIALLY_DEFERRED   INFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT ASSERTIONS_INITIALLY_DEFERRED_NOT_NULL NOT NULL
         CHECK_TIME  INFORMATION_SCHEMA.CHARACTER_DATA
               CONSTRAINT ASSERTIONS_CHECK_TIME_CHECK
         CHECK ( CHECK_TIME IN ('IMMEDIATE', 'DEFERRED' ) ),
         VALID_TIME_SUPPORT CHARACTER_DATA
               CONSTRAINT VALID_TIME_SUPPORT_CHECK
          CHECK (VALID_TIME_SUPPORT IN ('SEQUENCED','NONSEQUENCED','NONE')),
         VALID_TIME_PERIOD  INFORMATION_SCHEMA.CARDINAL_NUMBER,
         TRANSACTION_TIME_SUPPORT CHARACTER_DATA
               CONSTRAINT TRANSACTION_TIME_SUPPORT_CHECK
          CHECK (TRANSACTION_TIME_SUPPORT IN ('SEQUENCED','NONSEQUENCED','NONE')),
         TRANSACTION_TIME_PERIOD  INFORMATION_SCHEMA.CARDINAL_NUMBER,

         CONSTRAINT ASSERTIONS_PRIMARY_KEY
               PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_
         SCHEMA, CONSTRAINT_NAME ),

         CONSTRAINT ASSERTIONS_FOREIGN_KEY_CHECK_CONSTRAINTS
               FOREIGN KEY (CONSTRAINT_CATALOG, CONSTRAINT_
         SCHEMA, CONSTRAINT_NAME ) REFERENCES CHECK_CONSTRAINTS,

         CONSTRAINT ASSERTIONS_FOREIGN_KEY_SCHEMATA
               FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
          REFERENCES SCHEMATA,

         CONSTRAINT ASSERTIONS_DEFERRED_CHECK
               CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
            VALUES ( ( 'NO',  'NO' ),
                     ( 'YES', 'NO' ),
                     ( 'YES', 'YES' ) ) )
          )
```

Note to proposal reader: This adds two columns: TRANSACTION_TIME_SUPPORT and TRANSACTION_TIME_PERIOD.

19 Annex A (informative) Implementation-defined elements

1) Add the following item to the list of implementation-defined elements.

1) Subclause 3.1, “Definitions”: The transaction-time precision is implementation-defined.

20 Acknowledgments

This change proposal presents an improved and extended version of some of the constructs in TSQL2, which was designed by a committee consisting of Richard T. Snodgrass (chair), Ilsoo Ahn, Gad Ariav, Don S. Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T.Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo and Surynarayana M. Sripada. Their participation in the TSQL2 design was critical.

We thank Mike Sykes for help with the definition of transaction time.

The first author was supported in part by NSF grant ISI-9202244 and by grants from IBM, the AT&T Foundation, and DuPont. The second and third authors were supported in part by the Danish Natural Science Research Council, grant 9400911. In addition, the third author was supported by grants 11–1089–1 and 11–0061–1, also provided by the Danish Natural Science Research Council. The paper was produced in part during visits by the first author to Aalborg University and by the second author to the University of Arizona.