On the Semantics of 'Current-Time' In Temporal Databases

Marcelo Finger
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
Tel: +55 11 818 6135
Fax: +55 11 818 6134
email: mfinger@ime.usp.br

Peter McBrien
Dept. of Computer Science
King's College London
Strand
London WC2R 2LS
Tel: +44 171 873 2469
Fax: +44 171 873 2851
email: pjm@dcs.kcl.ac.uk

Abstract

The notion of the *current-time* is frequently found in work on temporal databases, and is usually accessed via a special interpreted variable called now. Whilst this variable has intuitive and simple semantics with respect to the processing of queries in temporal databases, the semantics of the variable inside transactions has not been studied. With the growing maturity of temporal databases, and the need to construct production quality temporal DBMS to demonstrate the usefulness of the technology, it is necessary that a complete study be made of transaction processing in temporal databases. This work aims to meet this objective by (1) detailing the alternatives that exist in the evaluation of now, (2) studying the problems of using now in current transaction processing systems, and (3) provide a formal framework for the processing of temporal transactions, and give rules in the framework for avoiding problems in evaluating now.

Keywords: Temporal databases, valid-time, transaction-time, transaction processing.

1 Introduction

The concept of a special interpreted variable called now is frequently found in work on temporal databases [WJL93, CDS+93, Sar90, Tan93], and models the intuitive notion of the current-time of a database operation. Despite its widespread use, no detailed consideration is ever given as to the exact method by which the value of now is evaluated, and what impact using now has on the transaction and concurrency control mechanisms used in a DBMS. The paper makes three contributions: (1) to give a detailed analysis of the choices available in the evaluation of now, (2) to demonstrate that the evaluation of now using standard transaction management rules may sometimes result in concurrent transaction executions which do not serialise, and (3) to define a formal framework with transaction management rules for the processing of temporal transactions, which allows the usage of now in serialisable transactions.

The paper is organised as follows. Section 2 defines the main concepts the paper builds upon: the main features of temporal databases, and the temporal properties of transactions. Section 3 discusses how now may be evaluated in a temporal database, and demonstrates the situations where non-serialisable transactions result. Section 4 presents a formal framework to analyse the semantics of temporal transaction execution; several distinct semantics are discussed, showing how the various possibilities for transaction control and the evaluation of now presented in Section 3 may be realised.

2 Preliminaries

We first present the important concepts of temporal databases as they impact on the transaction processing control to be used, and give a transaction processing framework tailored for temporal databases.

2.1 Time in databases

Many applications involving data with a temporal component may be represented in a framework [Jen94] which involves two temporal dimensions, namely the *valid-time* which reflects the periods of validity of data in the *universe of discourse* (UoD), and *transaction-time* which reflects the periods the data holds in the database. These two notions are orthogonal, so each one separately, or together, may be stored in a temporal database. A database that stores only the former is called a *valid-time database*, and one that stores only the latter is called a *transaction-time database*. A database that stores both it is a *bitemporal database*.

Databases that model only the temporal behaviour of the UoD might only incorporate valid-time information. However, [Fin94] notes that for the study of the dynamics of valid-time databases, the notion of real-time of execution (and hence the transaction-time associated to facts in the database) must always be taken into consideration. That is due to transaction-time being associated to updates by virtue of the serialisation ordering of transaction.

In our analysis, we will assume that the valid-time stored in a temporal database is a discrete linear flow of time, which is in accordance with the majority of the temporal databases in the literature [MS91, TCG+93]. The indivisible discrete unit of time —

the minimal interval — of a time dimension is called a *chronon* [Jen94]. In [CDS+93] a *clock tick* is the transaction-time corresponding to the border between two transaction-time chronons. Since transaction-time is determined by a measurement of real-time, this chronon length is also the effective granularity of real-time. For simplicity we will assume that that transaction-time and valid-time chronons are identical.

It is natural and common to have the notion of the 'current-time' [WJL93, CDS+93, Sar90, Tan93] in temporal databases, which is usually realised as a special variable called now. This notion of now appears to be similar to the *moving time* variable of [CW83]. However, in all the cited literature, the exact method by which the value of now is calculated is not stated, and this will be the subject in Section 3. We denote the use of the now variable as either *intentional* if it appears in the query statement, or *extensional* if it is embedded in the data. Temporal databases have been proposed which support both usages.

2.2 Properties of transactions

A fundamental property of transactions in DBMS is that they should obey the ACID properties [HR83]. An important part of maintaining the ACID properties in concurrent transaction execution is ensuring that the execution is *serialisable* [BHG87, Ull88]; which requires that when various database operations of different transactions are interleaved, the end result is the same as if the two transactions had been executed in *some* serial order.

Temporal databases bring an additionally requirement on this serialisation. In the definition of transaction-time in [Jen94] it is stated that the serialisation order of transactions should respect the transaction-time associated to the objects effected by the transaction. We will study transactions in a temporal framework where they are caused by *events* which can be:

- *external*: a user or an application program external to the database is submitting the transaction to the database.
- *internal*: generated by the processing of another transaction in the database. In general, internal events may be generated either by a transaction directly issuing a command to run a transaction, such as *split* or *spawn* [CR91], or indirectly by a transaction modifying a table, such at Sybase table triggers [MD92]. We reserve the term *cascaded transaction* to refer to those internally triggered transactions caused by a transaction that has yet to commit.
- *temporal*: the clock ticks of the internal system clock.

Whatever event causes a transaction, it may be in one of four states: *submitted*, *running*, *committed*, and *aborted*. These states are associated with certain real-times in the transactions life history:

- *submission-time*: the real-time associated to the (internal, temporal or external) event triggering the transaction.

- *begin-time*: the real-time at which the transaction becomes one of the running transactions in the system — in effect the time of the first entry in the log file.
- *commit-time*: the real-time representing the end of a transaction's execution, when the commit action is recorded in the transaction log [BHG87].
- *abort-time*: the real-time representing when the abort of the transaction was recorded in the transaction log.
- *lifespan*: the interval between begin-time and commit-time.

The properties described above are general, and no particular assumption has been made about the data model, though our presentation will be given in terms of the relational model. Note that the real-time associated to the commit-time of the transaction is by definition [Jen94] the transaction-time of the transaction, and methods to implement this definition in 2PL are presented in [LS93]. Indeed, we may regard the real-time axis as also being the transaction-time axis, since transaction-times are simply the real-times when a certain type of event occurs.

3 The Evaluation of now in Temporal Transactions

This section describes the alternatives that exists in evaluating the value of now in transactions, given that we follow the temporal framework described in Section 2.1.

3.1 Responsibility for Deciding the Current-Time

When performing an operation in a temporal database, we classify three methods of determining the value of now:

- now_t denotes a value determined by the real-time in which some event occurs. The determining event is normally transaction related, eg its submission, beginning or commitment.
- now_u denotes a user or application outside the DBMS supplying the value of use.
- now_r denotes that the value of the real-time clock when then operation is being performed is used.

It should be noted that the intentional usage of now_u is equivalent to simply embedding some explicit time in the query. Both now_t and now_u give a constant value of now for each transaction, whilst now_r allows the value to vary within a transaction.

3.2 User determined values of now

In Table 1 in we show a typical usage of now where we store records of employees and their monthly salary payments in EMPLOYEE, and represent the fact that Marcelo and Peter currently earn £2,000.

EMPLOYEE				
NAME	ID	SALARY	RANK	VALID-TIME
'Marcelo'	100	2000	'Researcher'	[1/1/1994,now]
'Tony'	101	2300	'Researcher'	[1/1/1994,6/8/1994]
'Peter'	102	1800	'Researcher'	[1/1/1989,4/4/1994]
'Peter'	102	2000	'Lecturer'	[5/4/1994,now]

Table 1: Contents of the EMPLOYEE relation

Our intention is to indicate by the use of now in the schema extension the semantics that a persons salary remains fixed until such time that a specific alteration is made. If we use now_u, we allow the user query to determine the period that a persons salary is at a certain level. Clearly for the extensional use of now, we will not want to use the now_u definition.

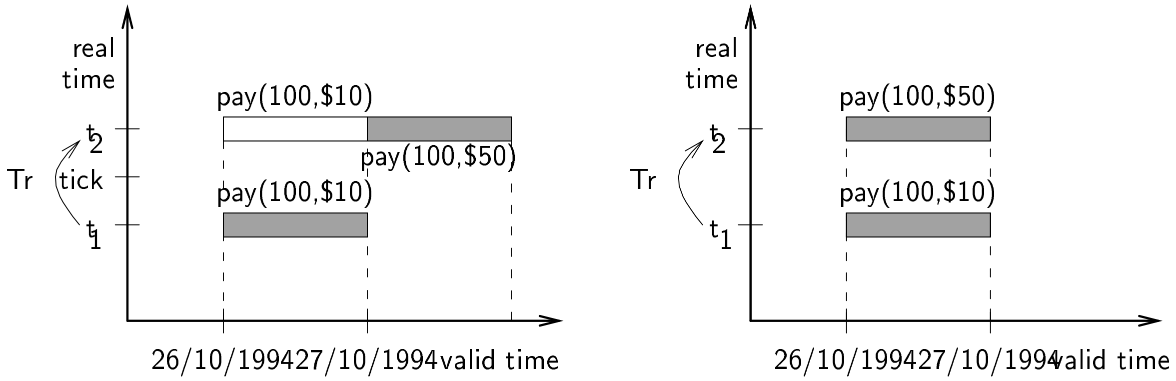
3.3 Real-time determined values of now

It is generally regarded that now should be constant for the period of a database operations [CDS+93], but since previous work has not considered in any detail the use of the variable in transactions, it is unclear whether now should be constant for the duration of a transaction. Using the real-time definition now_r would lead to the potential for the value of now to vary in transactions containing multiple operations.

The consequences of allowing now to change during a transaction are illustrated by the following example, where the interplay between the real-time and the valid-time of data is shown as a graph in Figure 1, and it is assumed that the chronon is one day. Suppose there is a transaction Tr that reads the value associated to payment p_1 at the now, multiplies it by 5, and updates it at the now again. Figure 1(b) illustrates the desired behaviour of such a transaction, with current-time 26/10/1994, starting at time t_1 and committing at t_2.

If a clock tick occurs during Tr, $t_1 < t_{tick} < t_2$, and if the transaction is allowed to 'see' such change, a different and undesirable result is obtained, as illustrated in Figure 1(a). This happens because Tr reads the value of the fact at one current-time when it starts execution during 26/10/1994, and updates the fact at is end, when execution is during 27/10/1994. Choosing a larger granularity for valid-time increases the likelihood of a transaction being completely executed within one chronon, but there may still be transactions which cross the border from one chronon to the next.

In order to avoid the interference of clock ticks with transaction processing, now must be seen as a constant for a given transaction, and thus now_r should not be used. Note that this rules out the use of the SQL92 CURRENT_TIMESTAMP variable [ISO92] as an implementation of now, since this varies with the value of the real-time clock within a transaction. This is not a surprising result, since CURRENT_TIMESTAMP was not designed to be used for the implementation of now in temporal databases. For the remainder of this paper we will assume that we choose to make now constant for any particular transaction. Note however, this does not prevent distinct concurrent transactions having distinct current-times.



(a) now varies (b) now constant

Figure 1: Varying or constant now

3.4 Event time determined values of now

The previous discussion has revealed that there are undesirable aspects to the semantics of both now_u and now_r, and so we now focus on the remaining option of now_t as the preferred choice. From the framework presented in Section 2.2, our choice is between submit, begin, and commit-times to determine the value of now. If we are using strict two-phase locking (2PL) concurrency control mechanism, we can not use commit-time to decide the value of now within the transaction, since its value would become known only after the transaction commits [LS93].

If begin-time is adopted, the value of now is not known at submission-time; in fact, since the delay between submission and execution is arbitrary, the value of now used in the processing of a transaction is an arbitrary value from the point of view of a user/program submitting a transaction. If submission-time is chosen, delays in entering the running state will not affect the current-time seen by a transaction. Either choice may lead to the effect described next.

3.5 Current-Time Moving Backwards

If submission-time determines the current-time seen by transactions, the value of now seen by a transaction about to begin may be less than that seen by a committed one. A serialisation of their execution would show the current-time 'moving backwards'. This situation is illustrated in Figure 2. Transaction T4 has now=26/10/1994, is submitted after transaction T3 which has now=25/10/1994, but T4 commits before T3 even begins its execution, hence they can only be serialised as [T4, T3].

Choosing begin-time to decide now might seem to solve this problem, but in fact only reduces it, since the serialisation of transactions is not dependent on their begin-time, unless we were to use preemptive locking with its prohibitive burden to transaction

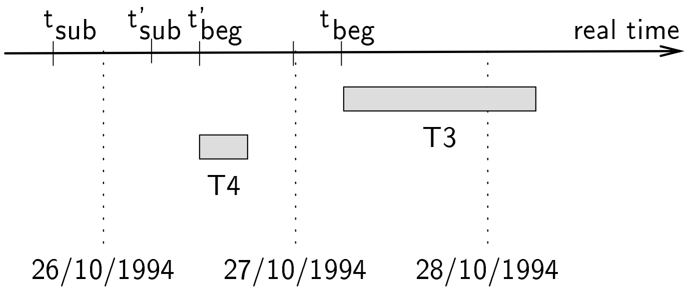


Figure 2: now moving backwards

throughput.

Having made this observation, there are two courses of action to take:

- View that now sometimes moving backwards is not a problem at all. Since valid-time database updates are allowed to alter present, past or future dates (w.r.t. the current real-time), the value of now not progressing in step with the serialisation order may not seem very important. Also, some transactions make no use of now, and process information with respect to a time built into the DML statement. Such a time can be in any order with the current transaction-time, and the system must be prepared to handle them anyway, so there is no problem in allowing a sequence of transactions where the order of current-times seen is not strict.

- View now moving backwards as breaking the serialisation rules. From the definition in [Jen94] that transaction-times of data obey the serialisation order of transactions, the fact that now is derived from the same real-time used to determine transaction-time should forbid the moving backwards effect. We should thus take some course of action to make submission or begin-time determine the serialisation order of transactions. Preemptive locking [BHG87, Ull88] would do this, we detail in Section 4 slightly less prescriptive strategies.

3.6 Serialisation of Transactions

In this sub-section we see that using begin-time to determine now leads to some concurrent transaction executions which are permitted by normal 2PL concurrency control rules [BHG87, Ull88], but which fail to produce results which are not serialisable.

In Table 1 we list the values for the EMPLOYEE relation, and Table 2(a) the values of PAYMENTS to those employees before execution of the following two transactions:

T5 Pay all current employees a bonus of £50

a. read EMPLOYEE: to find current employees

b. write PAYMENTS: to give each employee £50

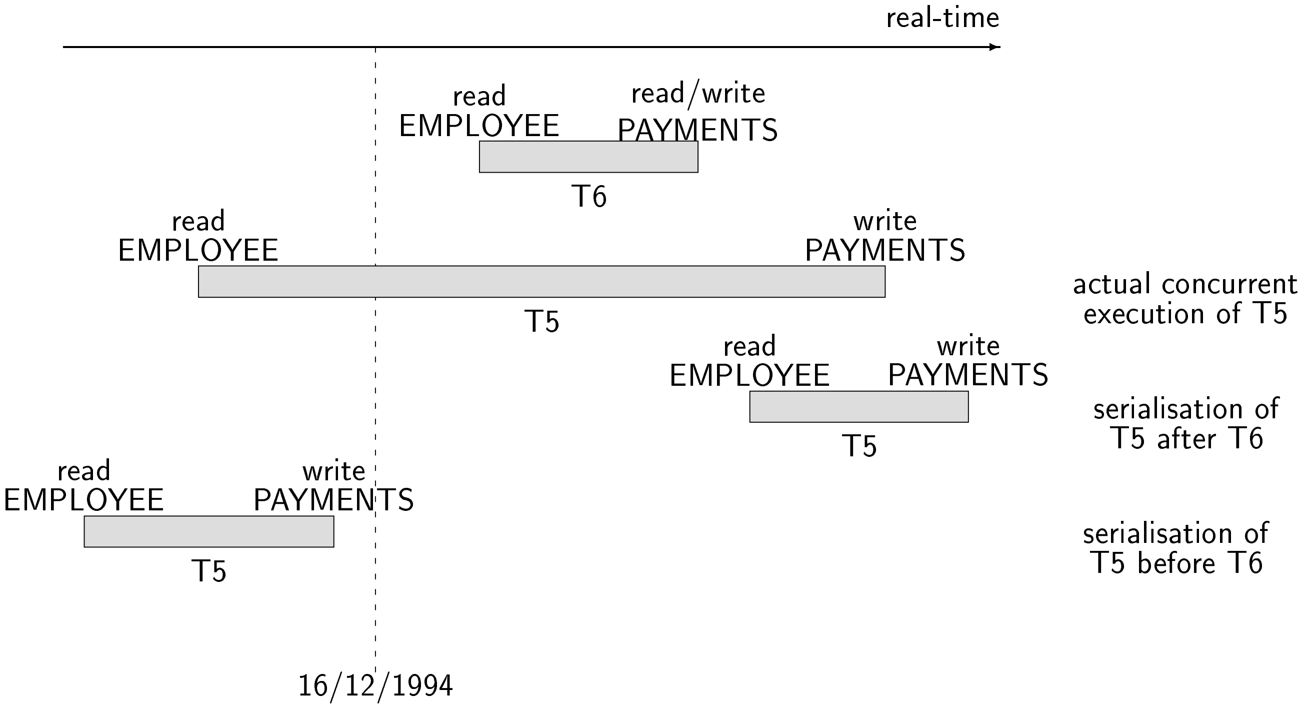


Figure 3: Execution of T5 and T6

T6 Pay all current employees who have earned less than £1000 so far this year a bonus of £100

a. read EMPLOYEE: to find current employees
b. read PAYMENTS: to find current employees with less than £1000
c. write PAYMENTS: to give each underpaid employee £100

For a standard 2PL system [BHG87, Ull88], both these transactions require a read lock on EMPLOYEE and a write lock on PAYMENTS. Thus when we have a concurrent execution of T5 and T6, the statements of each transaction interleaved, except the read T6b and write T6c must not get separated by the write T5b. Thus in Figure 3 we show three correct executions of T5 and T6. Taking an execution of T6 as a reference point, we show:

- an execution of T5 concurrent with T6. For the purposes of the example say this is what actually occurred.
- a serial execution of T5 after T6.
- a serial execution of T5 before T6.

Let us assume that we have adopted the begin-time of a transaction as the value of now. Given the ordering of actions shown in Figure 3 for the concurrent execution of T5 and T6, this would give the result in Table 2(c). However, if the serialisation was [T5,T6], then T6 would find that employee 100 had already been paid £1000 during the year, and thus give the result in Table 2(b). If the serialisation was [T6,T5], then the read of

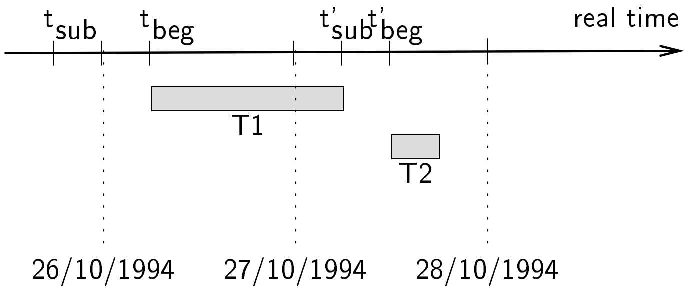


Figure 4: Internally triggered transactions

EMPLOYEE at the start of T6 must get a value of now equal or after that found by T5, and hence the result in Table 2(d) shows the two payments to 100 as occurring on the same date. Neither serialisation gives the same result as the concurrent execution, and thus we have violated the ACID properties of transaction execution. Thus we should not use begin-time as the value of now in a transaction with 2PL. We can look upon this being due to the fact that the value of now in a table is not subject to locking when real-time progresses, and thus 'moving' a transaction about in time (as is done when we serialise them) alters the semantics of execution.

If we were to adopt the submission-time as the value of now, then the result of the [T6,T5] serialisation would be Table 2(c) (assuming that the submission-time of T5 was 15/12/1994), i.e. the same as the result of the concurrent execution. We now have a serialisation of the transactions, and hence obey the ACID properties. The reason why submission-time works where begin-time does not is that the value of now does not change as we 'move' the transaction about in time during the serialisation process.

Note that the situation is to a degree unintuitive, due to the time moving backwards phenomenon. Having submitted T5 before T6, we might expect that its changes would be made first. This is not the case, since employee with ID=100 gets a low pay bonus, despite having recorded a salary of £1000 before those dates. This unintuitive behaviour can only be resolved by altering the basic method used to manage transactions, and we return to this topic in Section 4.

3.7 The Current-Time of Internally Triggered Transactions

The difficulty in maintaining serialisation of transactions will be increased when we permit transactions to be generated by internal events (whether cascaded or not). Consider a transaction T1 that is triggered by an external event, which at commit-time, triggers a transaction T2 via an internal event. A possible run of T1 and T2 illustrated in Figure 4.

The external event is submitted at real-time t_{sub}. Due to the load of the system, the execution of the triggered transaction T1 only begins at real-time t_{beg}, after a clock tick has occurred. T1 generates the internal event to trigger T2 at time t'_{sub}, which is also the commit-time of T1, and T2 starts execution at real-time t'_{beg}.

Either t_{sub} or t_{beg} determines the current-time seen by T1. Hence, it is clear that T1

PAYMENTS			
ID	AMOUNT	REASON	VALID-TIME
100	475	'Salary'	1/1/1994
100	475	'Salary'	1/6/1994
101	500	'Salary'	1/1/1994
102	500	'Salary'	1/1/1994
102	500	'Salary'	1/6/1994

(a) Before execution

PAYMENTS			
ID	AMOUNT	REASON	VALID-TIME
100	475	'Salary'	1/1/1994
100	475	'Salary'	1/6/1994
100	50	'Christmas Bonus'	15/12/1994
101	500	'Salary'	1/1/1994
102	500	'Salary'	1/1/1994
102	500	'Salary'	1/6/1994
102	50	'Christmas Bonus'	15/12/1994

(b) Execution of T5 followed by T6

PAYMENTS			
ID	AMOUNT	REASON	VALID-TIME
100	475	'Salary'	1/1/1994
100	475	'Salary'	1/6/1994
100	50	'Christmas Bonus'	15/12/1994
100	100	'Low Pay Bonus'	16/12/1994
101	500	'Salary'	1/1/1994
102	500	'Salary'	1/1/1994
102	500	'Salary'	1/6/1994
102	50	'Christmas Bonus'	15/12/1994

(c) After concurrent execution of T6 and T5

PAYMENTS			
ID	AMOUNT	REASON	VALID-TIME
100	475	'Salary'	1/1/1994
100	475	'Salary'	1/6/1994
100	50	'Christmas Bonus'	16/12/1994
100	100	'Low Pay Bonus'	16/12/1994
101	500	'Salary'	1/1/1994
102	500	'Salary'	1/1/1994
102	500	'Salary'	1/6/1994
102	50	'Christmas Bonus'	16/12/1994

(d) Execution of T6 followed by T5

Table 2: Contents of the PAYMENTS relation

and related T2 will not see the same current-time. The only condition enforced is that T2 sees a current-time greater or equal than that seen by T1. If equality of current-times is desired, a possible solution is to enforce that the event that triggers the related transaction T2 carries as a parameter the value of now, so that all queries and updates are evaluated relative to that time, ignoring the current real-time. Alternatively the system may provide a mechanism through which related transactions can automatically see the same value of now, which we detail in our formal model for temporal database transaction execution in Section 4. Both are uses of now_u.

3.8 Summary of Choices For Determining Now

We have discussed that now may be based on a user defined value, the real-time of the current operation or the real-time of some event in the transaction life history. We determined that the reasonable choice to make is to based the value on an event in the transaction life history. We have discussed that choosing either submission-time or begin-time as the determiner of now leads to the time moving backwards phenomenon, which breaks the definition of transaction-time in [Jen94] if strict 2PL is used. We have also shown that using begin-time may cause strict 2PL executions which fail to be serialisable. Finally, we discussed the possibility that internally triggered transactions may inherit the value of now from the triggering transaction. The next section will present a transaction execution model which removes the problems of time moving backwards, and shows how the use of submission time may be implemented.

4 A Temporal Transaction Execution Model

The choice of method by which transactions waiting to execute are selected for execution has a critical influence on the performance of a DBMS, but also, as seen in the previous section, it alters the temporal semantics of the transaction, since the value of the now variable depends on its evaluation in transaction-time. In this section we will present a formal framework for temporal transaction processing, review various possible execution strategies for transactions, and study their impact on the temporal semantics of the transaction execution.

Since we are interested only in the *temporal* semantics of transactions, we will omit details about the different semantics not affect the issue in question. In particular, when we model transaction execution in Figure 5, we show the impact on the temporal semantics (i.e. the value associated with now) as being the same if a transaction COMMITs or ROLLBACKs, by modelling them both as a general END. In this section, we shall ignore the effect of allowing *cascaded transactions*, where transactions that are running may start a new transaction executing before they have committed. Hence when we illustrate the transaction model in Figure 5, we show the execution of other transactions as being the result of the END of a transaction being reached. Section 5 extends the model to allow for cascaded transactions, and also introduces the notion of transactions triggered by temporal events.

The state of a DBMS which has an extended transaction model can be described by a tuple $\langle R, T, I, E\rangle$ where:

- R is the list of *running transactions*, currently being executed in the DBMS, that is they have passed BEGIN TRANSACTION[1] but not reached a COMMIT TRANSACTION or ROLLBACK TRANSACTION. The list is ordered by the transaction-time at which the BEGIN TRANSACTION was executed.

- S is the sequence of *submitted transactions* awaiting execution (that is the issuing of BEGIN TRANSACTION). In general, several entries in S may result from a single element in I or E. The list is ordered by the *transaction selector* which chooses which element of I or E should be placed in S, and where it should be placed in S.

- I is the list of *internally triggered transactions* which have yet to be selected for execution. The list is ordered on the transaction-time at which the transaction trigger was generated.

- E is the list of *externally triggered transactions* which have yet to be selected for execution, and is ordered by the transaction-time at which the request to execute the transaction was issued.

Each transaction t is represented by a tuple $t = \langle n, i, s \rangle$ where

- n is the name of the transaction, and thus identifies the code that defines the transaction

- i is the instance number of the transaction, and is unique to each invocation of the transaction

- s is the reference time of the transaction, and is what we use in evaluating now inside the transaction [CDS+93].

In defining executional semantics of transactions, we are interesting in defining transition rules of the form $\langle R, S, I, E \rangle \Rightarrow \langle R', S', I', E' \rangle$, where we perform manipulations on each list X to generate a new list X'. Any of the transaction lists R, S, I, E may be described by giving its elements $t_1, t_2, \ldots$ in a comma separated list inside square brackets, and we may concatenate lists together using the $\circ$ operator, such that $t_1 \circ [t_2, t_3] = [t_1, t_2, t_3]$. By abuse of notation, we will use $\circ$ on either single transactions or lists, such that $[t_1] \circ [t_2] = t_1 \circ [t_2] = [t_1] \circ t_2 = [t_1, t_2]$. For the execution of the model in Figure 5 to be fully specified, we need to specify at least one rule in each of the classes itemised below. In the following list we give with each class the transition rule that models the behaviour of a transaction manager for a typical DBMS, such as Sybase [MD92]. Note how in such systems the time associated with now varies according to how long the transaction is delayed in various buffers[2].

- *accept external transaction* (AET): remove an element of E and place it in S

$$\langle R, S, I, E \rangle, t \in E \quad \Rightarrow \quad \langle R, S \circ [C(t)], I, E - t \rangle \tag{1}$$

[1] which might be implicit if an ANSI/ISO model is used

[2] Indeed, it may even vary during the execution of the transaction.

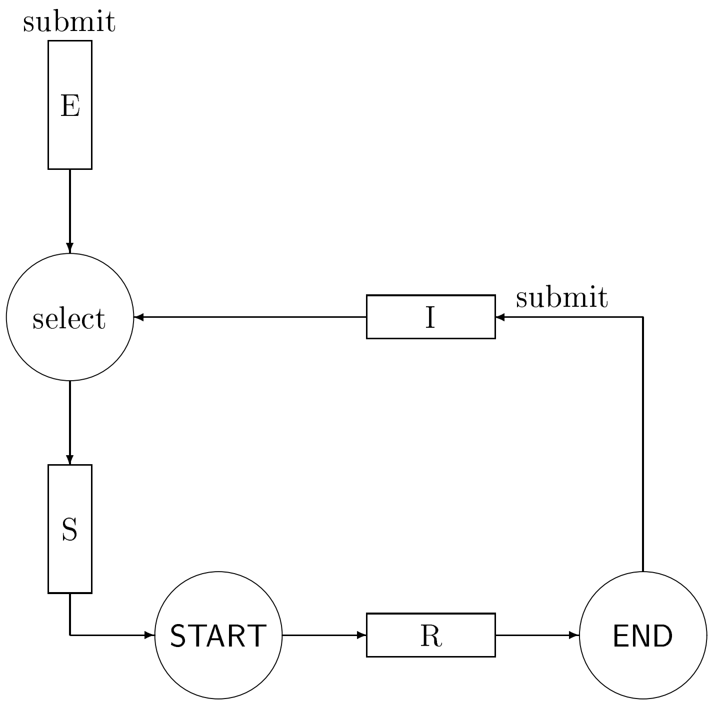


Figure 5: Interaction of transaction lists from $\langle R, S, I, E\rangle$

The C function is used to assert that the current transaction-time should be associated with the transaction, and has the definition $C(\langle n, i, s\rangle) = \langle n, i, \textsf{transaction-time}\rangle$. This corresponds to how standard DBMS handle transactions, in that asking what is the time always returns the current real-time clock, which we represent by the interpreted variable transaction-time.

- *accept internal transaction* (AIT): remove an element of I and place it in S

$$\langle R, S, I, E\rangle, t \in I \quad \Rightarrow \quad \langle R, S \circ [C(t)], I - t, E\rangle \tag{2}$$

- *start transaction* (ST): remove an element of S and place it in R

$$\langle R, S, I, E\rangle, t \in S \quad \Rightarrow \quad \langle R \circ [C(t)], S - t, I, E\rangle \tag{3}$$

- *end transaction* (ET): remove an element of R, and add to I all transactions $t_1, \ldots, t_n$ it has triggered

$$\langle R, S, I, E\rangle, t \in R \quad \Rightarrow \quad \langle R - t, S, I \circ [C(t_1), \ldots, C(t_n)], E\rangle \tag{4}$$

	Rule for			
	AET	AIT	ST	ET
Standard DBMS (e.g. Sybase)	(1)	(2)	(3)	(4)
Submission Time	(5)	(6)	(7)	(4)
Temporally Connected	(5)	(6)	(7)	(8)
External Time Preserving + Submission Time	(9)	(6)	(7)	(4)
External Time Preserving + Temporally Connected	(9)	(6)	(7)	(8)

Table 3: Summary of rules necessary to implement transaction semantics

4.1 Alternative Transaction Execution Semantics

We will now review various alternative execution strategies for our transaction model, paying attention to the impact of the evaluation of the now variable. We will do this by refining the general rules given above, so as to avoid some of the phenomena described in Section 3. Table 3 summarises the possible execution semantics for transactions which preserve the temporal ordering property.

4.1.1 Use Submission Time instead of Begin Time

The use of the submission-time of a transaction to determine the value of now has the advantage that the value does not vary arbitrarily as the delays in transaction execution vary. It is also essential if we are to have serialisable transactions when using nonconservative locking strategies. To achieve this involves altering the AET, AIT and ST rules to preserve the submission-time, giving the following versions of these rules:

$$\langle R, S, I, E\rangle, t \in E \;\Rightarrow\; \langle R, S \circ [t], I, E - t\rangle \tag{5}$$

$$\langle R, S, I, E\rangle, t \in I \;\Rightarrow\; \langle R, S \circ [t], I - t, E\rangle \tag{6}$$

$$\langle R, S, I, E\rangle, t \in S \;\Rightarrow\; \langle R \circ [t], S - t, I, E\rangle \tag{7}$$

4.1.2 Temporally Connected Transactions

To avoid the effect shown in Section 3.7 of internally triggered transactions evaluating now to different values, we must ensure that internally triggered transactions are given the submission-time of the external transaction which originally triggered them, possibly transitively via several other internally triggered transactions. This is achieved by modifying ET rule (4) as follows:

$$\langle R, S, I, E\rangle, \langle n, i, s\rangle \in R \;\Rightarrow\; \langle R - t, S, I \circ [\langle n_1, i_1, s\rangle, \ldots, \langle n_m, i_m, s\rangle], E\rangle \tag{8}$$

4.1.3 External Time Preserving

If we are concerned that transactions executing should not see the values of now moving backwards, as described in Section 3.5, then we must prevent the selection of externally

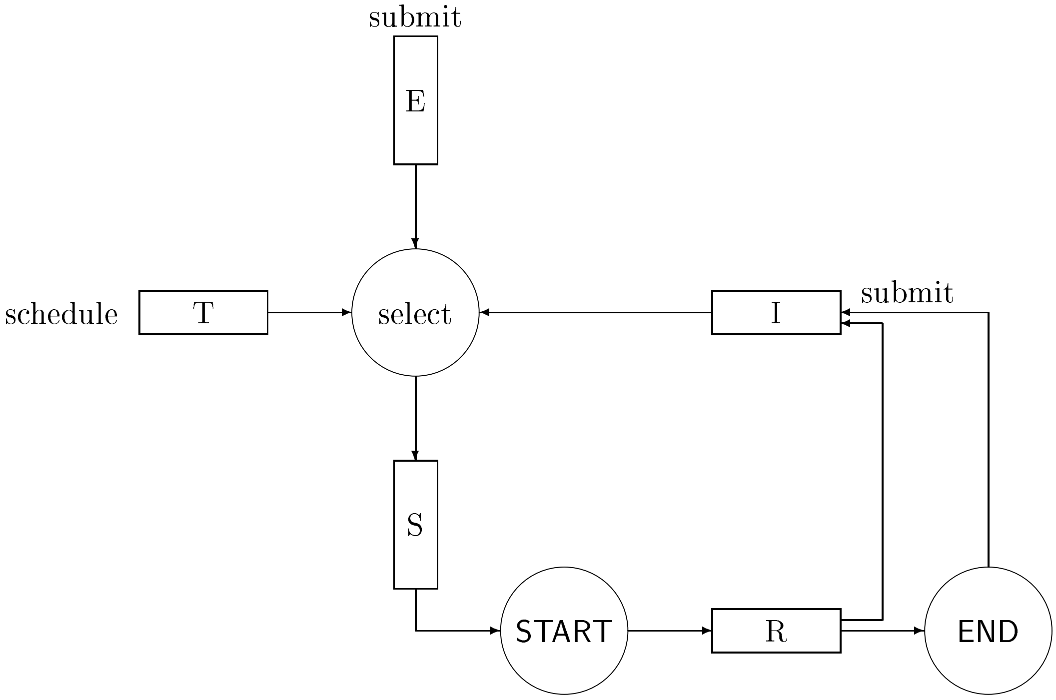


Figure 6: Interaction of transaction lists from $\langle R, S, T, I, E\rangle$

triggered transactions with a submission-time later than transactions already present in R, S, I. This amounts to providing the following AET rule:

$$\langle R, S, I, \langle n, i, s\rangle \circ E\rangle, \neg\exists n_r, i_r, s_r.(\langle n_r, i_r, s_r\rangle \in R \circ S \circ I), s_r < s \quad \Rightarrow \quad \langle R, S \circ \langle n, i, s\rangle, I, E\rangle \tag{9}$$

This semantics may be used in combination with either of the two previous semantics proposed.

5 Extensions to the Transaction Model

In this section we consider extensions to the executional model for temporal transactions in a database system which allow:

- the ability to trigger transactions of the basis of *temporal events*, by which we mean the current value of now reaching a particular moment in time.
- the ability to *cascade* transactions; a transaction may be triggered by another one which has yet to commit.

Both these enhancements require an alteration to the transaction execution model in Figure 5, to give the model shown in Figure 6. The nature of these alterations and the rules necessary to handle them are given in the following subsections.

5.1 Handling Temporal Events

We may model the transactions scheduled against temporal events in a similar to the E and I lists, as a list of scheduled transactions T. The submission-time associated with each transaction in the list is the time of the temporal event. If the temporal event is repeating (such as 'at the start of each day') then the transaction appears an infinite number of times in the list, each time associated with a recurrence of the event. A single new rule class is necessary to handle the input of transactions from the new list.

- *accept temporal event* (ATE): remove an element of T and place it in S

$$\langle R, S, T, I, E\rangle, t \in T \quad \Rightarrow \quad \langle R, S \circ [t], T - t, I, E\rangle \tag{10}$$

5.2 Cascaded Transactions

The handling of cascaded transaction manifests itself in Figure 6 as a link directly between the set of running transactions R and the set of internally triggered transactions I. The use of cascaded transactions will lead us to consider how *cascaded aborts* affect the temporal executional model. Thus two rule classes are required to handle cascaded transactions. The first handles the new link in the model, and the second performs the necessary task of removing all descendants of a transaction which has to be rolled-back.

- *cascade transaction* (CT): place an element in I, based on the current execution of one element of R

$$\langle R, S, T, I, E\rangle, t \in R \quad \Rightarrow \quad \langle R, S, T, I \circ [t_1, \ldots, t_n], E\rangle \tag{11}$$

- *rollback transaction* (RT): remove an element from R, and and all other elements from I, S, R which have been cascaded from it

$$\langle R, S, T, I, E\rangle, t \in R \quad \Rightarrow \quad \langle R - t - D(R, t), S - D(S, t), T, I - D(I, t), E\rangle \tag{12}$$

  where $D(L, t)$ is the set of transactions in list L which have been cascaded by transaction t.

6 Summary

We have reviewed the possible semantics that may be associated to the `now` interpreted variable. Firstly, we justified the implied definition found in the literature that the value of `now` should be derived from real-time inside the DBMS, and should remain constant for the lifespan of a transaction. We then showed that choosing either submit-time or begin-time for the value of `now` would lead to the time moving backwards phenomenon, which breaks the definition of transaction-time in [Jen94]. We also showed that choosing begin-time can lead to concurrent transaction executions which fail to serialise. Problems are also found when internally triggered transactions are permitted.

We presented a formal framework for the definition of transaction processing rules to be used by a transaction manager in a temporal DBMS. The framework was used to propose transaction processing strategies which solve the problems with using now that we have described.

Further work is needed to: (a) determine which of the proposed semantics for the evaluation of now is most appropriate in general, or whether a temporal database should make a range of possibilities available to be chosen by transactions; (b) prove that the execution rules result in a correct implementation; and (c) study how the handling of time relates to classical serialisation theory; it may be the case that serialisation theory needs to be extended to support some of the semantics of now presented here. The latter point would mean moving the solution (to the problems of time moving backwards and lack of serialisation) from an external layer that can be attached to a normal DBMS into the concurrency control mechanism. That constitutes a serious change to the design of current DBMS specifically to support the handling of time, so this issue should be discussed with greater detail than what space allows us here. Preliminary results on this work can be found in [FM95].

Acknowledgments

The early stages of the work described within this paper was partially funded by the EU under the TEMPORA project number E2469 in the Esprit programme. The authors would like to thank members of the project for much help and advice during the project.

References

[BHG87] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[CDS+93] J. Clifford, C. Dyreson, R.T. Snodgrass, T. Isakowit, and C.S. Jensen. Now in TSQL2. Technical report, The TSQL2 Language Design Committee, December 1993.

[CR91] P.K. Chrysanthis and K. Ramamritham. A formalism for extended transaction models. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 103–112, Barcelona, Spain, 1991.

[CW83] J. Clifford and D.S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 6(2):123–147, 1983.

[Fin94] M. Finger. *Changing the Past: Database Applications of Two-dimensional Temporal Logics*. PhD thesis, Imperial College, Department of Computing, February 1994.

[FM95] M. Finger and P. McBrien. Concurrency control for transactions in valid-time databases. Technical Report 95-07, King's College London, 1995.

[HR83] T. Härder and A. Reuter. Principles of transaction-oriented database recovery. *Computing Surveys*, 15(4), 1983.

[ISO92] ISO/IEC. Database language SQL (SQL-92 or SQL2). Technical Report 9075:1992, ISO/IEC, 1992.

[Jen94] C.S. Jensen *et al.* A consensus glossary of temporal database concepts. *SIGMOD Record*, 23(1):52–64, 1994.

[LS93] D. Lomet and B. Salzberg. Transaction-time databases. In Tansel et al. [TCG+93], chapter 16, pages 388–417.

[MD92] D. McGoveran and C.J. Date. *Sybase and SQL Server.* Addison Wesley, 1992.

[MS91] L.E. McKenzie and R. T. Snodgrass. Evaluation of relational algebra incorporating the time dimension in databases. *ACM Computing Surveys*, 23(4):501–544, December 1991.

[Sar90] N. Sarda. Algebra and query language for a historical data model. *Computer Journal*, 22(1):11–18, 1990.

[Tan93] A.U. Tansel. A generalized relational framework for modeling temporal data. In Tansel et al. [TCG+93], chapter 7, pages 183–201.

[TCG+93] A.U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design and Implementation.* Benjamin/Cummings, 1993.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[WJL93] G. Wiederhold, S. Jajodia, and W. Litwin. Integrating temporal data in a heterogenous environment. In Tansel et al. [TCG+93], chapter 22, pages 563–579.