An Alternate Approach to Time Stamping Temporal Data
Jay Koleszar

ABSTRACT
When there is a requirement to capture history in a table, it becomes necessary to add a date, and perhaps the time, to a tuple to establish when it was valid. The classical method of working with temporal data is to use starting and ending timestamps to mark this period. This works well for recording and querying the data, but it makes the data processing more complex. The requirement to maintain data integrity requires the ending timestamp of one record must equal the starting timestamp of the next. Failure to do so will create duplicate records, or gaps in the history the records are trying to capture. This paper will examine a single timestamp model and show that it is sufficient to meet the needs of temporal data. A number of requirements that any model of temporal data must satisfy are set forth, and a comparison between these two models is made for each of these requirements. Showing how each of these models meet the requirements illustrates the advantages and disadvantages of each. Though not a formal requirement, transaction processing will also be examined. The degree of data integrity maintained is a direct result of how well transaction processing is conducted. The single timestamp model is a simpler model. Querying this model is more complex without the redundant second timestamp, but the processing steps are significantly easier, and therefore the resulting data integrity should be higher.

Keywords

1. INTRODUCTION
Temporal data is part of many databases. In some applications it is only data, such as an employee’s birthday. In this instance it is a fact that happens to be a date; but it is treated no differently than any other fact about an employee, such as address, phone number, or height. In other applications, it is an integral part of a tuple, stating the period during which this tuple is valid. Data used in this manner creates a historical table. Richard Snodgrass defines a Valid-Time State Table as, “It records information valid at some time in the modeled reality and it records states, that is, facts that are true over some period of time. The From-Date and To-Date columns determine the ‘period of validity’ of the information in the row.” [4] There are two types of temporal data that may be stored in a table. The current fashion is to describe them as Valid Time and Transaction Time. Valid Time is the period during which the facts were true. Transaction Time is the period when the tuple itself was valid. Both ends of these periods are recorded with dates and perhaps times, depending on the granularity needed to accurately describe the data. The term timestamp is often used to describe the boundaries of these periods. A table that contains both valid times and transaction times is said to be bi-temporal.

With non-temporal tables, data can be changed, by inserting, modifying, or deleting records as appropriate. All records reflect the current state of the database. When the history of a table is to be maintained, then a timestamp must be added to indicate when data and/or the record itself became effective. Now the only operations that can be allowed are insertion and updating. Deleting a record destroys data and with it, the history. In order to record a deletion without destroying a record, a second timestamp is usually introduced. The period of validity is the time between a starting and an ending timestamp. This approach is sound and has been used for many years. It is the one most often discussed in the literature, but there are a number of problems implementing this design. Some of these include:

- Determining what value to use for an ending timestamp when the value is unknown. What is known is this date may happen at some point later in the future, but not before the present time. This is known in the field as the Now problem and it is still being discussed in the current literature. [1] [5] [6]
- Data integrity issues that will arise if there are gaps or overlaps between the ending timestamp of a record and the starting timestamp of the subsequent record. Code must be implemented to prevent this from happening.
- Needing to record the ending timestamp of the previous record, when that time will not be known until the transactions of appending a new record with the corresponding timestamp, modifying the old record and checking data integrity has been completed. [4]

All of these issues have been addressed in numerous papers, and many solutions have been presented; but there is an alternative approach. Only a single timestamp is required for temporal records. In fact, the second timestamp is redundant when it contains the same information as the starting timestamp on the next record. Many historical tables have been created by simply appending new data, along with a timestamp of when it was added to the table, or by copying the updated record to a historical table. The current record is the one with the latest timestamp. This is a simpler approach and has some obvious advantages. Only using one timestamp could save six, eight, or more bytes per record; depending on how the data type used for the timestamp, is stored. This storage requirement increases with each timestamp used. With a single timestamp, all modification can be made by appending a new record. There is no ending timestamp that needs to be recorded, so existing records never need to be modified. Transaction processing is simpler and the database should be more responsive without the need for record locking required for updating.

The single timestamp model appears to be a simpler model with some advantages. The question is whether or not this model will meet the needs of temporal databases. This paper will show that it is sufficient to meet all of the requirements and that this model is easier to implement. In the following paragraphs the model will be described in more detail. Then to determine if this model is sufficient for temporal databases, it will be tested against a set of
2. PROPOSED MODEL
The main feature of this Single Timestamp Model (STM) is the inclusion of a single date/time field to record the starting date/time of an event to be recorded. This could be for a valid time event, or a transaction event, or both. The current record is the one with the latest timestamp. The length of time or period that an event was valid is from the timestamp on a record until the timestamp on the next record in sequence. When the starting timestamp is the current record, then the end of the period is the current date/time on the system. In this model, modifications to existing records are prohibited. The only way to modify a record is to append a new record, with the appropriate timestamps.

In some temporal databases there is a need to be able to terminate valid time events. For instance Joe got fired. This is the end of Joe’s employment. In this case there must be some kind of flag to indicate the end of an event. This could be done by changing a field to some set value indicating a terminating event. Joe’s position could be set to “Fired” and the timestamp would be set to indicate when this was effective. Another approach could be to add an additional field containing a logical value. In the classic Two Timestamp Model (TTM), the ending date is used to terminate a record. In order to do this, Joe’s current record would have to be modified to replace the current value for the ending date with the date Joe was fired. If this were a bi-temporal table, then the transaction ending date would also have to be modified with the timestamp corresponding to when this event was recorded in the database. The problem with using an ending timestamp to indicate the termination of a record is that doing so requires an update of the current record, and this negates one of the advantages of the STM. In this model we will use an additional logical field to indicate the termination of a valid time event.

For transactions, a single timestamp is sufficient. Records are only appended, so it is only necessary to record when this event happened. A record is considered valid until it is replaced by a new record with a later timestamp. There is no reason to terminate a record. The facts recorded in a tuple can cease to be valid at some point in time, but the record recording this will be valid forever.

3. REQUIREMENTS FOR WORKING WITH TEMPORAL DATA
While the STM is a simpler approach, the question remains, can this approach meet all of the needs of temporal data. Richard Snodgrass developed a series of requirements for accessing and setting constraints on temporal data in [4]. This provides a good starting point for developing a list of the requirements that a temporal model must meet. These requirements include the ability to query the data, or the state of the database, at any point in time. There must also be the ability to enforce constraints on the data to prevent queries from returning duplicate records, and to maintain referential integrity. Finally there must also be methods to indicate the end of valid events, and to correct errors in data entry.

3.1 Querying Temporal Data
There are three classes of queries that can be made on temporal data. They can be summarized as:

- Non-temporal – Display the current record.
- Non-sequenced – Display what was at any point in time.
- Sequenced – Display the history over time.

Queries must be able to return the correct results for both valid and transaction type temporal data.

3.2 Constraints on Temporal Data
There must be constraints on temporal data such that there are no duplicate values at any point in time, and continuous records cannot have gaps in time. These data integrity issues can be characterized as:

- Value Equivalent Duplicates – All the fields are equivalent except the timestamp.
- Non-sequenced Duplicates – All the fields are equivalent including the timestamp.
- Sequenced Duplicates – Value equivalents and periods of overlapping time.
- Current Duplicates – A special case of Sequenced Duplicates where the period of overlap involves Now.
- Gaps – When the ending of one record is prior to the start of the next record and thus creates a period during which there is no valid data.

The existence of any of these conditions could cause queries to return erroneous results.

3.3 Constraints to Enforce Referential Integrity
The requirement of Referential Integrity is another type of constraint. It requires that each value in a foreign key field of the referencing table must match a key value in the referenced table. For temporal tables, this constraint adds another level of complexity; as now this relationship must be true at each point in time. For as long as a tuple containing a foreign key is valid in the referencing table, the corresponding key value must also exist at least for the same period. It is also possible to have a combination of temporal and non-temporal tables related to each other. Each creates their own unique requirements as shown in Table 1. Referential integrity is a key requirement of all databases for maintaining data integrity.
Table 1: Relational Requirements of Temporal Tables

<table>
<thead>
<tr>
<th>Requesting Table</th>
<th>Requested Table</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreign Key</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>Non-Temporal</td>
<td>Non-Temporal</td>
<td>Nothing new from current database design</td>
</tr>
<tr>
<td>Temporal</td>
<td>Non-Temporal</td>
<td>Nothing new from current database design</td>
</tr>
<tr>
<td>Non-Temporal</td>
<td>Temporal</td>
<td>A current value of the Key must exist for every Foreign Key.</td>
</tr>
<tr>
<td>Temporal</td>
<td>Temporal</td>
<td>The period for the Key must be greater or equal to the period of the Foreign Key.</td>
</tr>
</tbody>
</table>

3.4 Ending Temporal Records and Correcting Errors

There are a couple of additional requirements that the STM must meet. They are the ending of temporal records and error correction. These are not an issue with the TTM, as both of these requirements are easily satisfied using an ending timestamp. When working with valid times, many records deal with objects that will go on forever. For these a single timestamp works well to record the history of changes to the object. When records deal with relationships that do not go on forever, a temporal database must have some method to record the end. An example of this is Joe got fired. A related issue is the requirement that data errors can be corrected in a manner such that queries of temporal data yield the correct results, whenever a query against this data is made.

4. MODEL SATISFIES REQUIREMENTS

The requirements for a temporal data model have been set in the paragraphs above. Now the test is, to determine if the STM can satisfy these requirements. The following paragraphs will show that it can. In addition a comparison will be made between the two models to show the advantages of each. To look at these issues, the same data set that was used in [6] (shown in Table 2) will be used.

Table 2: Sample Data Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Started</th>
<th>Recorded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Shoe</td>
<td>1/1/04</td>
<td>1/1/04</td>
</tr>
<tr>
<td>Bob</td>
<td>Outdoor</td>
<td>1/11/04</td>
<td>1/2/04</td>
</tr>
<tr>
<td>Jim</td>
<td>Toy</td>
<td>1/12/04</td>
<td>1/2/04</td>
</tr>
<tr>
<td>Jill</td>
<td>Shoe</td>
<td>1/19/04</td>
<td>1/2/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Outdoor</td>
<td>1/8/04</td>
<td>1/23/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Shoe</td>
<td>1/31/04</td>
<td>1/31/04</td>
</tr>
</tbody>
</table>

4.1 Querying Temporal Data

We can ask “What is the current department for all of the employees?” The basic technique employed with the STM is to use a sub-query or view to find the record with the lowest timestamp larger that the desired value. This timestamp then becomes a pointer to the desired record for display.

Code Block 1: Current Status

```sql
SELECT E1.Name, E1.Dept, E1.Started
FROM Employee E1
WHERE E1.Started =
    (SELECT Max(E2.Started)
    FROM Employee E2
    WHERE E1.Name = E2.Name
    GROUP BY E2.Name);
```

This query returns one record for each of the employees and the date they started in their current Department (the record with the most recent Start Date). The data returned from this type of query appears the same as it would from a non-temporal table where the data is updated by changing the appropriate values. The status for any point in the past can be determined by adding a single additional constraint (highlighted in bold) to the above query.

Code Block 2: Previous Status

```sql
SELECT E1.Name, E1.Dept, E1.Started
FROM Employee AS E1
WHERE E1.Started =
    (SELECT Max(E2.Started)
    FROM Employee E2
    WHERE E1.Name = E2.Name
    AND E2.Started <= #1/11/04#
    GROUP BY E2.Name);
```

This query returns two records showing that on the 11th of January, there were two employees, both working in the Outdoor Department. Displaying history (Sequenced Query) is done by removing the restrictions on the timestamps and adding the temporal date to the ORDER BY clause of an SQL statement.

Code Block 3: Show History

```sql
SELECT E1.Name, E1.Dept, E1.Started
FROM Employee AS E1
ORDER BY E1.Name, E1.Dept, E1.Started
```

Queries involving transactional times can be handled in exactly the same manner as was shown above for valid data. The same type of queries can be used in the TTM. For the TTM, the sub-queries are replaced with constraints on the starting and ending dates greater than or equal to the specified point in time. The TTM has an advantage when an important element of the tuple is the period between the start and ending date. In these cases we may care more about how long a person was in a position, than...
about when. Queries to return information about these periods will be faster for the TSM than the STM. It is possible to simulate the TTM for tables with only a single timestamp by creating a view for the non-current records and then joining this view with the original table. The following is an example:

**4.2 Constraints on Temporal Data**

In traditional tables one of the constraints that is established is that there should by no duplicate records. This is often done with the use of a primary key. In temporal tables we need to have the additional constraint that duplicate values, at any point in time, should not be allowed. With a single timestamp, there are only two types of duplicates that can occur, non-sequenced and sequenced. Non-sequenced duplicates (having the same data and the same timestamp) can be prevented by adding the timestamp to the primary key. Sequenced duplicates (where the data is the same, but the timestamps are different) do not present a problem in this alternate approach; as when the data is queried, only one record will be returned. True, there is an unnecessary record in the table, and a query might not return the earliest start date. Removing this record can be easily done by coalescing the data. Coalescing is the merging of overlapping intervals for value equivalent tuples. This coalescing can be done at any time [1], but might be best done at the time of updates.

Sequenced and Current Duplicates, and Gaps are only a problem with the TTM. They can occur because the ending timestamp of one record does not correspond to the starting timestamp of the next record. A Sequenced Duplicate will exist when the ending timestamp of one record is later than the starting timestamp of the second record. Duplicate records will exist for the period between the start date of the second record and the ending date of the first. Current Duplicates are a special case of Sequenced Duplicates that exists when one or both of the ending timestamps are Now. These duplicates can be prevented by the use of “triggers” to test for duplicates before they are appended to a table.

Gaps in the records can occur when the starting timestamp is later than the ending timestamp for the previous record. This could be intentional, but where records are intended to be continuous, the database programmer must take steps to insure the ending and starting timestamps correspond to each other. This is a clear example where the TSM requires far more processing than the STM. Steps must be taken to insure the ending and starting timestamps are correct.

### 4.3 Constraints to Enforce Referential Integrity

The job of insuring referential integrity is done when data is updated. Again the STM has advantages. In this model the only type of update permitted is appending records. Before updating the referencing tables, the program must ensure that any foreign keys in this new record have a matching key in the referenced table and that its record is current. For the referenced table, termination is the only activity which might impact the referential integrity. Therefore when terminating a key, a search must be made of only the current records of the referencing tables to determine that there are no longer any active foreign keys referencing it before the terminating record is appended.

With the TTM these updates are more complex. Now both the starting and the ending timestamps must be compatible. This means that in addition to testing the start dates, the ending dates must also be tested. The period for which the key is valid must be greater than or equal to the period the foreign key is valid. There also has to be a consideration for when the ending date on the key in the referenced table has a value other than Now. If the referencing table is temporal, then its ending date can be set to the ending date for the key. If the referencing table is non-temporal, then there is a problem. The foreign key in a non-temporal table is assumed to be valid forever, but it references a key that at some point in time may no longer be valid. This is a violation of referential integrity.

A related issue for the TTM is that the join algorithms commonly found in relational databases are not designed to efficiently support non-equi-joins [2]. Joining two temporal tables requires adding lines like “AND T1.Start <= T2.End AND T1.End >= T2.Start” to the WHERE clause of the SQL statement. The STM uses sub-queries which contain only equijoins.

### 4.4 Ending Temporal Records and Error Correction

There are a number of ways to terminate records. In the traditional temporal database with starting and ending timestamps, the date Joe was fired is added to the ending timestamp for his current position. This could be the best solution if the database

---

**Table 3: Results from the view, vEndedJobs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Started</th>
<th>Ended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Shoe</td>
<td>1/1/04</td>
<td>1/7/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Outdoor</td>
<td>1/8/04</td>
<td>1/30/04</td>
</tr>
</tbody>
</table>

**Table 4: Results from the combined queries**

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Started</th>
<th>Ended</th>
<th>Recorded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Outdoor</td>
<td>1/1/04</td>
<td>1/2/04</td>
<td></td>
</tr>
<tr>
<td>Jill</td>
<td>Shoe</td>
<td>1/19/04</td>
<td>1/2/04</td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td>Toy</td>
<td>1/12/04</td>
<td>1/2/04</td>
<td></td>
</tr>
<tr>
<td>Joe</td>
<td>Shoe</td>
<td>1/1/04</td>
<td>1/7/04</td>
<td>1/1/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Outdoor</td>
<td>1/8/04</td>
<td>1/30/04</td>
<td>1/23/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Shoe</td>
<td>1/31/04</td>
<td>1/31/04</td>
<td></td>
</tr>
</tbody>
</table>

---
has to support a large number of queries dealing with the intervals or periods. If this is not a requirement, then just adding a new record with Joe’s position recorded as “Fired” with a timestamp recording the event, should suffice. A third approach would be to add a flag field to the tuple to indicate the timestamp is the end of a valid time event. This method would have to be used for those relationships where multiple values are permitted, e.g. Joe is allowed to hold several positions at once. Now in the STM, when querying for active records, we have to add the condition where the position is not “Fired”, or the termination flag is false.

The question that naturally arises is; do transaction times also need some method of ending records? This paper maintains that they do not. Databases can be considered to “go on” for ever. It is not likely we will ever care how long a record was active. We do care when it was active. As was shown above, the state of the database can be determined at any point in time. When there are starting and ending timestamps for a record, then for sequential records the ending timestamp must match the starting one of the next record. Otherwise we will have duplicates or gaps in our records, neither of which is acceptable. Preventing this from happening during transaction processing is a complex task. Relationships may end and then be re-established at a later time, with a resulting gap. There are no valid reasons for wanting to have a gap when a record was not valid.

When the decision is made to have an ending date, it must also be decided how to represent an unknown date. This is a problem that has been with temporal database designers from the beginning, and is still being discussed in the current literature [1] [5] [6]. Solutions include the use of null, the earliest or latest system date possible, or some other arbitrary date. All of them can be made to work, and all of them have problems. The advantage of the proposed STM model is that this is not an issue. The most likely date to be unknown is an ending date. In this model an ending date is not used, so how to represent an unknown date is not an issue. The dates the model uses are a date when an event or relationship is started and the transaction timestamp when this is recorded in the database. Both of these will have known values.

Any valid scheme for handling temporal data must include a method of correcting data entry errors. Again this approach handles error correction in a natural fashion, though we do have to include an additional field to indicate this is the end of a valid time record. Let’s take for example than on the 10th it was recorded that Joe was transferred to the Toy Department. Later on the 23rd it discovered that Joe should have been transferred to the Outdoor Department instead of the Toy Department. The way to correct this error is to make two entries. The first is to end Joe’s stay with the Toy Department on the day that he started, and then add an additional record making the correct entry. The table for Joe’s employment history is displayed below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Started</th>
<th>End Flag</th>
<th>Recorded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Shoe</td>
<td>1/1/04</td>
<td>False</td>
<td>1/1/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Toy</td>
<td>1/8/04</td>
<td>False</td>
<td>1/10/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Toy</td>
<td>1/8/04</td>
<td>True</td>
<td>1/23/04</td>
</tr>
<tr>
<td>Joe</td>
<td>Outdoor</td>
<td>1/8/04</td>
<td>False</td>
<td>1/23/04</td>
</tr>
</tbody>
</table>

The scheme works, but when it is queried, will it return the proper results? We can obtain the correct results, but we will have to modify the previous queries to take into account the ending flag, and the transaction timestamp. To determine the state of the database on the 30th, the following query could be used:

```
Code Block 6: State of the database
FROM Employee AS E1
WHERE E1.EndFlag = False
AND E1.Recorded = (SELECT Max(E2.Recorded)
FROM Employee E2
WHERE E1.Name = E2.Name
AND E1.EndFlag = E2.EndFlag
AND E2.Recorded <= #1/30/04#
GROUP BY E2.Name, E2.EndFlag);
```

Expanding on this query, we could determine the current departments for all of the employees.

```
Code Block 7: Current Employee’s Department
FROM Employee AS E1
WHERE E1.EndFlag = False
AND E1.Recorded = (SELECT Max(E2.Recorded)
FROM Employee E2
WHERE E1.Name = E2.Name
AND E1.EndFlag = E2.EndFlag
AND E2.Recorded <= #1/30/04#
GROUP BY E2.Name, E2.EndFlag)
AND E1.Started = (SELECT Max(E2.Started)
FROM Employee E2
WHERE E1.Name = E2.Name
AND E1.EndFlag = E2.EndFlag
AND E2.Started <= #1/30/04#
GROUP BY E2.Name, E2.EndFlag);
```

5. MODEL FULFILLS ALL THE REQUIREMENTS
This model meets all of the requirements for handling Temporal Data. The data can be queried for both past and current states of valid time and the state of the database. Duplicate records can be prevented through the use of indexes. Referential integrity can be maintained between temporal tables by verifying the values in key fields exist before appending a record. Valid events and data corrections can be made with the use of an Ending Flag. This is another valid model. The issue is, not so much, “is this a better model?” Better is a relative term, and depends very much on the application. It is a simpler model, and one that fits naturally for tables where all data changes are made by appending records. There is never a need to modify existing records.

6. THE MODEL SOLVES CURRENT PROBLEMS IN TEMPORAL DATABASES
One of the biggest problems with the TTM is that the starting timestamp of one record must correspond to the ending timestamp
of the previous record. This means when records are updated, one record must be modified, and a new record appended to the table. This usually applies to transaction timestamps, but the same issues can arise with valid time timestamps when they are assigned by the system. If this could be done instantaneously, there wouldn’t be any problems; but this is the real world and this processing does take time. The amount of time taken can be significant, depending on the amount of processing required and the amount of database activity. The central issue is that the value for the ending and beginning timestamps will not be known until all of the processing has been completed and the transactions committed. But, in order to process the records, some value must be entered. If this is not done carefully, it opens up the possibility of time slice query being executed, while the table is being updated, that would return results that violate the consistency requirements of the database. The requirements for transaction processing are set forth in [3] and are known by the acronym, “ACID”. Atomicity: A transaction is atomic; either all or nothing. Consistency: Actions taken as a group do not violate any integrity constraints. Isolation: All transactions have the appearance of being executed one at a time. Durability: Upon successfully committing a transaction, the resulting changes will survive failures. Some of these issues, and possible solutions, are outlined in [6].

With the proposed STM, all of these processing problems are avoided. There is no need, and in fact it is not allowed, to modify any previous record. The new record will not be added until all of the processing is completed and the transaction committed. At this time, the timestamp value is known and can be entered into the table. All the requirements of ACID are satisfied. More importantly they are satisfied without a lot of complex transaction processing.

7. SUMMARY
This paper has shown that a single timestamp is sufficient for working with both valid and transaction time temporal data. This is a simpler model that is well suited to working with a large class of temporal data that has been created by appending every change to an existing table. This model satisfies the requirements of querying temporal data, imposes the necessary constraints to prevent duplicate records, and maintains referential integrity. All updating of data can be done by appending new records. Existing records are never modified.

The technique used to query temporal data with a single timestamp is to use a sub-query or view, to find a data record at some point in time. The sub-query or view finds the maximum timestamp value to determine the current record, or the latest timestamp prior to a desired value for the active record at any point in the past. In essence, this technique finds the correct timestamp, which is unique and thus can serve as a pointer to the desired record.

The querying of data may well be slower with this STM because it requires the use of sub-queries or views to do what the redundant ending timestamp does. This is offset by a number of advantages. Constraints are easier to enforce. Duplicate records can be prevented by simply adding the timestamp to the primary key. Gaps in records will only happen if they are deliberately inserted, and cannot be created by a failure of processing. Transaction processing is easier to perform, as only new records need to be appended. There is no need to synchronize the ending timestamp of the old record with the starting timestamp of the new.

Finally, it will be up to the database designer to decide if the ease of data processing offsets the cost of query response time. Remember, fast queries with poor data integrity are no substitute for slower queries with the correct results.

REFERENCES