Literature Review: Realising Temporal Information Modelling

Stephan Maree
University of Cape Town
Cape Town, Western Cape, South Africa
mrxste013@myuct.ac.za

ABSTRACT

This literature review aims to explore the Trend Temporal Conceptual Data Modelling Language and the challenges in implementing it in Database Management Systems. Fundamental topics in temporal data such as valid time and transaction time are discussed alongside Trend's extentions to the Enhanced Entity-Relationship Diagram. The implementations of temporal data in Database Management Systems like MariaDB and PostgreSQL are examined. Algorithms for converting both temporal and atemporal Entity-Relationship Diagrams are discussed, showing that some refinement is needed to suit them to Trend. A critical analysis of the temporal data implementations of various Database Management Systems is done to determine the best platform for implementing Trend.

1 INTRODUCTION

Temporal databases are gaining increasing popularity. There are many applications in which being able to view historical data and enforce rules about what can and can‘t happen to entities is vital. Examples of this are accounting applications that need to track when payments were made, and legal applications where verifying the validity period of laws is vital [21]. In an atemporal DBMS, these features need to be implemented by developers in their applications, this can be expensive and time-consuming. If temporal constraints were managed by the DBMS, these problems would be mitigated.

With the increasing popularty of temporal databases comes the need to effectively model the constraints of working with Temporal Information Systems For atemporal relational databases, Entity Relationship Diagrams (ER diagrams) are the standard for modelling the relationships and interactions of entities [8, 23]. Many Temporal Conceptual Data Modelling Languages (TCDML) have been proposed as extensions to ER diagrams, all with their respective strengths and shortcomings [3, 9, 22]. This project will focus on one particular TCDML, Trend [12]. Unlike most other TCDMLs, Trend has undergone multiple usability tests and has shown promise in being an accessible option for novice modellers [4].

The proposed project is designing a tool to convert a Trend model into a relational schema that enforces the various temporal constraints specified by the modeller. This will be using some existing standard for implementing temporal data into a Database Management System (DBMS) which will need to determined in this review.

The aim of this literature review is to investigate the additional features represented in Trend, explore the different ways temporal data is implemented in current DBMSs, and discuss various algorithms which convert both temporal and atemporal ER diagrams into relational schemas. This is followed by a comparison of the standards and approaches introduced.

2 BACKGROUND ON TEMPORAL DATA

This section discusses some of the important concepts in temporal data. Temporal databases typically have a discrete, linear time domain [5]. In a discrete time domain, time is broken into a set of units called *chronons*. Depending on the particular database, these could be seconds, days, or even years. Every event in the database is considered to occur for the entirety of the chronon it is assigned to and cannot be divided into smaller periods. Time is considered linear, as it moves in a single direction step-by-step.

Another concept popular in TDBMSs is the idea of *valid time* and *transaction time* [2, 13, 16, 22]. Valid time refers to the period in which the fact being modelled is considered true in the real world. Transaction time refers to the period in which the fact is represented in the database. If an employee was promoted on the 15th of March, but this was only entered in the database on the 18th, the valid time and transaction time would differ. Some TCDMLs, like *TimeERplus* further introduce *user-defined time* and *lifespan* [9]. However, these have not seen wide adoption in DBMSs [5, 13, 16]. A table that implements both valid time and transaction time is referred to as a *bitemporal table*.

An entity can change state in two different ways, it can either *evolve* into another entity and lose its original type, or *extend* and become a member of a new type while still being a member of its original type [2–4].

Time is represented in temporal databases using timestamps. A timestamp is a value that associates a tuple with a time value. This value could either represent valid time or transaction time. There are three types of timestamping [5, 14]:

- Time points
- Time intervals
- Set of time intervals

In a time points-based system, each tuple is assigned a single time point. The fact is assumed to be true for the entirety of that chronon. If a fact is true for multiple time points, multiple tuples are used to represent this. An example of a system for booking rooms in an office for meetings is shown in Table 1.

Table 1: Time point timestamping

BookNo	Name	RoomNo	Time
1	James	A1	1
1	James	A1	2
2	John	A1	3
3	Jane	A2	3
3	Jane	A2	4
3	Jane	A2	5
4	James	A1	6
4	James	A1	7

An additional "BookNo" attribute is added to be able to differentiate between one booking lasting four time units and four consecutive bookings.

Time interval timestamping removes the need of this additional attribute by representing the period of an event. This removes the need for explicitly differentiating between multiple consecutive and one long booking, as the longer booking would only be used in one booking. Disjoint periods are represented in different tuples. The data in Table 1 is represented using time intervals in Table 2.

Table 2: Time interval timestamping

Name	RoomNo	Time
James	A1	[1,2]
John	A1	[3,3]
Jane	A2	[3,5]
James	A1	[6,7]

Using a set of time intervals, sometimes referred to as a *temporal element* [5], represents the full history of a fact in a single tuple. This removes the ability to distinguish between multiple consecutive rentals and one long one, but comes with the benefit of being much more spatially efficient. Table 3 shows how the scenario from Table 1 is represented using time interval sets.

Table 3: Set of Time Intervals

Name	RoomNo	Time
James	A1	[1,2]∪ [6,7]
John	A1	[3,3]
Jane	A2	[3,5]

3 FEATURES PRESENT IN TREND

Trend is a temporal extension of the *Enhanced Entity-Relationship Diagram* (EER Diagram), which the reader is assumed to be familiar with. This means it comes equipped with the standard features of EER Diagrams [8, 23]: Entities, Relationships, Attributes, Cardinality Constraints, Inheritance, Specialization, and Generalization. This section discusses the additional features that Trend has and would need to be enforced in the project application [4, 12].

3.1 Temporal Elements

Since Trend allows the user to model both temporal and atemporal elements, it is important to distinguish between these. Trend gives the option to add a clock item to an entity, relationship, or attribute. This indicates that the respective element is temporal and can change. Entities and Relationships of a type do not always need to remain of that type and attributes do not always need to have a value. Attributes with a pin icon cannot change value once set.

3.2 Dynamic Extension and Evolution

Entities and relationships are able to change over time, this is implemented in Trend using *dynamic extension* and *dynamic evolution*. Dynamic extension is modelled with an arrow labelled with ‘EXT‘. This means that the source item can also become an item of the target type. Similarly, dynamic evolution is modelled with an arrow labelled with ‘CHG‘. This means that that the source item transforms into an item of the target type. A transition is mandatory if the line is solid, and optional if it is dashed. Having ‘CHG‘ and ‘EXT‘ be uppercase means that the transition occurs in the future, while lowercase refers to the past.

3.3 Quantitative Transition Constraints

By appending a number to the ‘CHG‘ or ‘EXT‘ label, one enforces the duration constraint for the specified number of chronons. A solid arrow from entity A to B labelled ‘chg 4‘ means that an entity of type A needed to have been a member of A for 4 chronons before changing its type to B.

4 TEMPORAL DATA MODELS

4.1 SQL:2011

In 2011, SQL introduced a new standard to support temporal data [13]. SQL:2011 implements temporal data using time periods. The time periods include the start time and goes up to, but excluding, the end time. It introduced the notion of *Application-time periods* and *System-time periods*, which is equivalent to valid time and transaction time, as discussed in Section 2. We summarise the features introduced in SQL:2011 [13].

Application-period tables can be made by creating a table containing a period with a user-defined name, for example:

```
CREATE TABLE students(
StuID INTEGER,
StuName TINYTEXT,
Dept TINYTEXT,
StartTime DATE,
EndTime DATE,
PERIOD FOR stuPeriod(StartTime,EndTime)
)
```

A tuple can be loaded in as follows:

```
INSERT INTO students
VALUES (1001, "Janene", "Math", ‘2024-01-01‘, ‘2027-01-01‘)
```

This will create the following table:

Table 4: An application-time table

StuID	StuName	Dept	StartTime	EndTime
1001	Janene	Math	2024-01-01	2027-01-01

If the student has changed their department from Math to Engineering in 2025, only to go back in 2026, we can update the table as follows:

```
UPDATE students
FOR PORTION OF stuPeriod
FROM DATE ‘2025-01-01‘ TO DATE ‘2026-01-01‘
SET Dept = "Engineering"
WHERE StuID = 1001
```

This will split the original row into three, resulting in the following table:

Table 5: The application-time table after an update

StuID	StuName	Dept	StartTime	EndTime
1001	Janene	Math	2024-01-01	2025-01-01
1001	Janene	Engineering	2025-01-01	2026-01-01
1001	Janene	Math	2026-01-01	2027-01-01

Deleting works similar to updating, as the user can specify a portion for deletion and the table gets split up according to the new existing periods.

Using this approach requires a new definition of the primary key. Table 5 shows how the StuID attribute is an invalid choice for the primary key, as it appears multiple times in the table. SQL:2011 allows for the triple (StuID,StartTime,EndTime) to be the primary key for the table. If the user wishes to prevent overlapping periods for the same StuID (For instance, if the student can only belong to one department at a time), they can specify this with the following command:

```
ALTER TABLE students
ADD PRIMARY KEY (StuID,stuPeriod WITHOUT OVERLAPS)
```

New referential integrity constraints were also added, preventing the referencing of entities at invalid times. Assume that there are two tables, named students and departments respectively:

Table 6: An example that does not have referential integrity

StuID	StuName	Dept	StartTime	EndTime
1001	Janene	Math	2024-01-01	2025-01-01
1001	Janene	Engineering	2025-01-01	2026-01-01

Dept	StartTime	EndTime
Math	2025-01-01	2027-01-01
Engineering	2024-01-01	2026-01-01

Here, it can be seen that Janene is a member of the Math department before it existed. This situation can be prevented using the following command:

```
ALTER TABLE students
ADD FOREIGN KEY (Dept,PERIOD stuPeriod)
REFERENCES departments (Dept, PERIOD depPeriod)
```

System-time tables work in a similar way, storing an attribute with the start time and the end time. The user is unable to modify the values for the start time and end time directly, only the DBMS has access to that. The user can create a System-time table for the previous example as follows:

```
CREATE TABLE students(
StuID INTEGER,
StuName TINYTEXT,
Dept TINYTEXT,
Sys_start TIMESTAMP(12) GENERATE ALWAYS AS ROW START,
Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
PERIOD FOR SYSTEM_TIME(Sys_start,Sys_end)
) WITH SYSTEM_VERSIONING
```

Data can be inserted as follows:

```
INSERT INTO students
VALUES (1001,"Janene","Math")
```

This results in the following table:

Table 7: A system-time table

StuID	StuName	Dept	Sys_Start	Sys_End
1001	Janene	Math	2024-01-01 15:01:32	9999-12-31 23:59:59

Update and delete only operate on the latest version of an entry. They both change the 'Sys_End' field of the latest entry to the current time; Update creates a new entry, while delete only modifies the end time.

System-time tables are simpler than application-time tables as only the current version of a row need to be considered, meaning that no extra referential integrity constraints need to be considered.

4.2 TSQL2

TSQL2 is a temporal extension of SQL proposed in 1994 [21]. An attempt was made to incorporate TSQL2 into the SQL standard, but this ultimately failed [13]. TSQL2 supports transaction time tables, valid time tables, and bitemporal tables. In an effort to make TSQL2 work with existing SQL databases, valid and transaction time columns are hidden. Temporal columns are nameless columns that contain a period datatype [6]. This decision led to the introduction of new syntax for accessing the hidden columns. Additionally, some of the operations do not have a clear specification, allowing for multiple possible outputs for the same query [6]. For the reasons mentioned, TSQL2 has received substantial criticism [6].

4.3 Temporal Database Management Systems

There are many Database Management Systems available with temporal support. This subsection will discuss how some of them have implemented temporal data.

- *MariaDB*: MariaDB fully implements SQL:2011, providing support for both system-time and application-time tables. Application-time tables are supported as of version 10.4.3 [17] System-time tables were implemented in 10.3.4, and provide a simpler syntax than proposed in SQL:2011, only requiring that the user include `WITH SYSTEM_VERSIONING` to create one. [18]
- *PostgreSQL*: PostgreSQL does not natively support temporal data. Extensions do exist that implement system-time tables, but none that implement application-time periods [20] The user would need to implement application-time tables manually.
- *TDSQL*: Tencent Distributed SQL follows the standards of SQL:2011, with some modifications to query processing that greatly improve performance [16].
- *Teradata*: Teradata does not follow SQL:2011, rather implementing the specification proposed in TSQL2 [21]. Temporal support was implemented as of release 13.10 and includes both valid time and transaction time support [1].

Other DBMSs like Oracle [19] and IBM Db2 [11] have implemented temporal support, but due to the lack of documentation on their implementation they have not been discussed in this literature review. NoSQL DBMSs that support temporal data, for example graph databases [7], are outside the scope of this review as they are not appropriate for the relational model of Trend.

5 ALGORITHMS FOR EXTRACTING RELATIONAL SCHEMAS FROM ER DIAGRAMS

When it comes to converting a TCDML like Trend into a relational schema, there are two approaches [9]:

- Map the model directly into a relational schema.
- Map the model into a conventional ER diagram first, and proceed to map that to the relational schema.

Since Trend is an extension of ER modelling, we will be focusing on the direct conversion. This section will first discuss an algorithm for mapping a conventional ERD into a schema. Afterwards it will discuss some proposed temporal algorithms.

5.1 Converting ERDs into a relational schemas

The traditional approach for mapping ER Diagrams into relational schemas goes as follows [8, 15]:

(1) *Map regular entity types.* For each entity, create a table that includes the attributes. If the attributes are composite, only include the simple component attributes. Select the primary key as specified in the diagram.
(2) *Map weak entity types.* Similarly to the previous step, create a table for each weak entity type. This time, include both the key of the weak entity and the owner as the primary key.
(3) *Map one-to-one relationships.* Choose one of the entities and add its primary key as a foreign key in the other. It is best to include the foreign key in the entity with total participation in the relationship. Any attributes assigned to the relationship should be included in the foreign key holder.
(4) *Map one-to-many relationships.* Include the key of the entity on the ‘one‘ side of the relationship as a foreign key to the entity on the ‘many‘ side. Just like before, any attributes assigned to the relationship should be included in the foreign key holder
(5) *Map many-to-many relationships.* Create a new table for the relationship. Have the combination of the keys of both participating entities be the primary key of the relationship table. Add any attributes of the relationship to the relationship table.
(6) *Map multivalued attributes.* Create a new table for the attribute. Have the combination of the entity‘s key and the individual attribute‘s key be the primary key of the new table. If the attribute is multi-valued, include the simple component attributes in the new table.
(7) *Map N-ary relationships.* When there have N entities participating in the same relationship, create a new table. Include the foreign keys of all participating entities as part of the new table‘s primary key. If any of the entities have a cardinality constraint of 1 to the relationship, their foreign key does not need to be part of the primary key.
(8) *Map specialization and generalization.* Create a table for each subclass. Include simple attributes and the primary key of its superclass. The primary key of this entity is the primary key of its superclass.

5.2 Temporal Conversion

The direct approach, as proposed in [10, 22], follows the same approach as in the previous section with an added step for enforcing the temporal constraints. These papers use *TimeERplus* and *ERT* instead of Trend, however their methods can be easily adapted to suit the project. In this section, we will summarise the algorithm presented in [22]. It should be noted that this paper uses the notion of complex object classes, which acts as an abstraction of a group of entities/values. The algorithm proposed goes as follows:

(1) For each entity (complex or simple), create a table and select a primary key. For every functionally dependent and time invarying role, add the key of the corresponding entity or value as an attribute.
(2) For each entity which is a subclass of another entity, replace its primary key with the primary key of its superclass.
(3) For each complex value class, create a table and add a primary key. Add each simple object to the table and add the primary key of each complex value type that is contained by it.
(4) For each entity (complex or simple), for each functionally dependent and time varying role, create a new table with the entity’s key and a period. Have the entity’s key be the primary key of the new table.
(5) For each relationship, create a new table. If the relationship is time varying, add a period column. Have the composite of the participating entities’ keys be the primary key of the relation.
(6) For each complex entity class, create a table of all the entities it includes. Add their keys as the attributes of the table. For every complex entity included in this new table, create a separate table containing the keys of each object and a period.
(7) Add additional temporal constraints defined by the ERT model. The details of this step have been omitted from this review as it is not relevant to Trend.

6 DISCUSSION

The amount of the temporal support in DBMSs vary substantially, with some of the differences shown in Table 8. Many of DBMSs, like PostgreSQL, only support transaction time, and not valid time [20]. Valid time tables are necessary for the project as they are able to enforce some of the fundamental temporal constraints required by Trend. Since SQL:2011 enforces both transaction and valid time, the DBMS used in the project would need to have full implementation of it, or an equivalent standard like TSQL2 [21].

SQL:2011 has received criticism for not including a PERIOD datatype [14]. The PERIOD datatype introduces useful commands like CONTAINS, EQUALS, PRECEDES, and OVERLAPS. Another feature that SQL:2011 has received criticism for not supporting is data coalescing. Data coalescing refers to combining tuples with overlapping periods together. Native support for this could be beneficial depending on the application.

Since these features mostly cater to convenience and are not strictly necessary, the framework proposed in SQL:2011 fulfills the criteria of the project. As shown in Table 8, MariaDB, TDSQL, and Teradata look promising for the proposed software. Out of these, MariaDB‘s simplified syntax and accessibility make it the ideal DBMS to use.

None of the DBMSs include support for the temporal transition constraints used in Trend. These need to be added manually using triggers, as mentioned in the section discussing the algorithms for converting the TCDML into a relational schema.

Table 8: A Comparison of the temporal implementations for different DBMSs

DBMS	Schema Specification	Valid Time	Transaction Time	Additional Features	Accessibility
MariaDB	SQL:2011	Supported	Supported	Simplified syntax for system-time tables	Open-source and free
PostgreSQL	None	Not Supported	Need extensions	None	Open-source and free
TDSQL	SQL:2011	Supported	Supported	High performance queries	Neither open-source or free
Teradata	TSQL2	Supported	Supported	None	Neither open-source or free

Another design problem worth considering is the use of system-time tables. Trend doesn‘t have any explicit use of transaction time in the models, so an option would be to simply not include them. Alternatively, every temporal entity could be made into both a system-time and application-time table, allowing the user to access all of the associated features. System-time tables are storage-intensive [16], so unnecessarily using them should be avoided.

The algorithm proposed in [22] and discussed in section 5.2 will need to be modified to be suited for Trend as it does not have an explicit notion of complex object classes, and the algorithm doesn't include temporal transition constraints.

7 CONCLUSIONS

Temporal databases have received increasing support over the last few years since the introduction of SQL:2011. As a consequence of this new growth, Trend was proposed as a TCDML which emphasised usability and expressiveness. This leaves room for a tool that is able to convert a Trend model directly into the schema of a DBMS.

This review discussed some of the fundamental concepts of temporal data, namely valid time and transaction time, entity transformations, and timestamping methods. We then discussed Trend's implementation of temporal elements, dynamic extension and evolution, and quantitative transition constraints.

This review further explored how temporal data is implemented in temporal DBMSs, highlighting standards such as SQL:2011 and TSQL2. The implementations vary across different DBMSs, some implement comprehensive support for temporal data, others offer very limited implementations of specific features. This necessitates careful consideration when selecting a platform for implementing Trend models.

Additionally, various existing algorithms for converting temporal/atemporal ER diagrams into relational schemas were summarised. This highlighted the need to refine these algorithms to facilitate the practical implementation of Trend.

REFERENCES

[1] Mohammed Al-Kateb, Ahmad Ghazal, Alain Crolotte, Ramesh Bhashyam, Jaiprakash Chimanchode, and Sai Pavan Pakala. 2013. Temporal query processing in Teradata. In *Proceedings of the 16th International Conference on Extending Database Technology* (Genoa, Italy) *(EDBT '13)*. Association for Computing Machinery, New York, NY, USA, 573–578. https://doi.org/10.1145/2452376.2452443

[2] Alessandro Artale and Enrico Franconi. 1999. Temporal ER Modeling with Description Logics. In *Conceptual Modeling — ER '99*, Jacky Akoka, Mokrane Bouzeghoub, Isabelle Comyn-Wattiau, and Elisabeth Métais (Eds.). Springer, Berlin, Heidelberg, 81–95.

[3] Alessandro Artale, Christine Parent, and Stefano Spaccapietra. 2007. Evolving objects in temporal information systems. *Annals of Mathematics and Artificial Intelligence* 50, 1 (2007), 5–38.

[4] S Berman, C Maria Keet, and Tamindran Shunmugam. 2024. The temporal conceptual data modelling language Trend. *Under review at Data Intelligence* (2024).

[5] Michael H. Böhlen, Anton Dignös, Johann Gamper, and Christian S. Jensen. 2018. Temporal Data Management – An Overview. In *Business Intelligence and Big Data*, Esteban Zimányi (Ed.). Springer International Publishing, Cham, 51–83.

[6] Hugh Darwen and C.J. Date. 2006. *An Overview and Analysis of Proposals Based on the TSQL2 Approach.* Apress. http://www.dcs.warwick.ac.uk/~hugh/TTM/OnTSQL2.pdf

[7] Ariel Debrouvier, Eliseo Parodi, Matías Perazzo, Valeria Soliani, and Alejandro Vaisman. 2021. A model and query language for temporal graph databases. *The VLDB Journal* 30, 5 (2021), 825–858.

[8] R. Elmasri and S.B. Navathe. 2004. *Fundamentals of Database Systems.* Addison-Wesley.

[9] Heidi Gregersen. 2005. Timeer plus: A temporal eer model supporting schema changes. In *British National Conference on Databases*. Springer, Springer, Berlin, Heidelberg, 41–59.

[10] Heidi Gregersen, L. Mark, and Christian Søndergaard Jensen. 1998. *Mapping Temporal ER Diagrams to Relational Schemas.* Number TR-39 in Technical Report. Aalborg Universitetsforlag.

[11] IBM. 2024. *IBM Db2.* https://www.ibm.com/products/db2

[12] C Maria Keet and Sonia Berman. 2017. Determining the preferred representation of temporal constraints in conceptual models. In *Conceptual Modeling: 36th International Conference, ER 2017, Valencia, Spain, November 6–9, 2017, Proceedings 36 (Lecture Notes in Computer Science, Vol. 10650)*. Springer, Cham, 437–450.

[13] Krishna Kulkarni and Jan-Eike Michels. 2012. Temporal features in SQL: 2011. *ACM Sigmod Record* 41, 3 (2012), 34–43.

[14] Florian Künzner and Dušan Petković. 2015. A Comparison of Different Forms of Temporal Data Management. In *Beyond Databases, Architectures and Structures*, Stanisław Kozielski, Dariusz Mrozek, Paweł Kasprowski, Bożena Małysiak-Mrozek, and Daniel Kostrzewa (Eds.). Springer International Publishing, Cham, 92–106.

[15] Vincent S Lai, Jean-Pierre Kuilboer, and Jan L Guynes. 1994. Temporal databases: model design and commercialization prospects. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems* 25, 3 (1994), 6–18.

[16] Wei Lu, Zhanhao Zhao, Xiaoyu Wang, Haixiang Li, Zhenmiao Zhang, Zhiyu Shui, Sheng Ye, Anqun Pan, and Xiaoyong Du. 2019. A lightweight and efficient temporal database management system in TDSQL. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2035–2046.

[17] MariaDB. 2024. *Application-Time Periods.* https://mariadb.com/kb/en/application-time-periods/

[18] MariaDB. 2024. *System-Versioned Tables.* https://mariadb.com/kb/en/system-versioned-tables/

[19] Oracle. 2024. *Databases.* https://www.oracle.com/database/

[20] PostGreSQL. 2024. *Temporal Extensions.* https://wiki.postgresql.org/wiki/Temporal_Extensions#:~:text=Postgres%20can%20be%20extended%20to,to%20be%20altered%20and%20queried.

[21] R Snodgrass. 1995. *The TSQL2 Temporal Query Language*. Springer, New York, NY.

[22] C Theodoulidis, Pericles Loucopoulos, and Benkt Wangler. 1991. A conceptual modelling formalism for temporal database applications. *Information Systems* 16, 4 (1991), 401–416.

[23] Jeffrey D. Ullman. 1988. *Principles Of Database And Knowledge-Base Systems*. Computer Science Press.