TRENDy: a tool for temporal modelling and database generation

Stephan Maree[1,†], Richard Taylor[1,†] and C. Maria Keet[1,*]

[1]Department of Computer Science, University of Cape Town, South Africa

Abstract

Temporal ontologies, knowledge graphs, and conceptual data models persist as requirements, since much data collected is temporal and constraints must hold to ensure data quality. Temporal conceptual modelling and ontology development face various challenges in uptake, notably the cognitive load and then extra efforts to squeeze it into an atemporal OWL, and once represented, to convert it into something concretely useful in applications. We introduce the TRENDy tool, which offers browser-based diagrammatic modelling of the logic-based temporal conceptual data modelling language Trend, and subsequent automated conversion into SQL statements to create the corresponding relational database. The evaluation showed the tool to outperform the baseline modelling tool in usability.

Software availability: https://github.com/richietaylor/trendy

Keywords

Temporal ontologies, Temporal conceptual models, Graphical modelling tool, Model transformations, SQL

1. Introduction

The visual paradigm for ontology authoring is well-known, and for conceptual data models such as UML and EER, it is considered essential. For temporal ontologies and conceptual models, this is even more important, because the extra set of constraints impose a higher cognitive demand, especially when ternary relations have to be reified in OWL [1, 2]. In addition, reasoning over temporal logics is computationally costly [3] and the transformation to temporal databases to gain a practical concrete advantage with the temporal constraints has not been designed, let alone implemented, although there are theoretical advances on temporal OBDA with a few constraints surveyed in [4] and on transforming temporal entity types to the relational model [5]. A transformation-based approach to link ontologies to data has been shown to be promising [6] thanks to its increased expressiveness, but only for atemporal application ontologies and EER models.

To address the temporal modelling and deployment hurdles, we developed the TRENDy tool. It provides, first, a graphical editor front end, which uses the syntax and semantics of the Trend modelling language [7, 8] that has a mapping into the Description Logic $\mathcal{DLR}_{\mathcal{US}}$ [9] that is is based on linear temporal logic with the Until and Since operators. It not only contains temporal entity types, relationships, and attributes, but also transitions constraints, principally among entity types and among relationships. For instance, to be able to assert that "each alumnus must have been a student before" and "an employee may also become a manager". Trend was extensively tested with over 1000 participants and shown to be effective for both understanding and modelling [8]. The TRENDy tool offers the first implementation of Trend.

Second, we designed and implemented transformation algorithms from Trend models into SQL such that all constraints are preserved. Effectively, columns for time are added and then triggers to ensure data integrity. This is, to the best of our knowledge, the first implementation of a conversion from a temporal conceptual model into a relational database.

EKAW 2024: EKAW 2024 Workshops, Tutorials, Posters and Demos, 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2024), November 26-28, 2024, Amsterdam, The Netherlands.

*Corresponding author.

†These authors contributed equally.

MRXSTE013@myuct.ac.za (S. Maree); TYLRIC007@myuct.ac.za (R. Taylor); mkeet@cs.uct.ac.za (C. M. Keet)

0000-0002-8281-0853 (C. M. Keet)

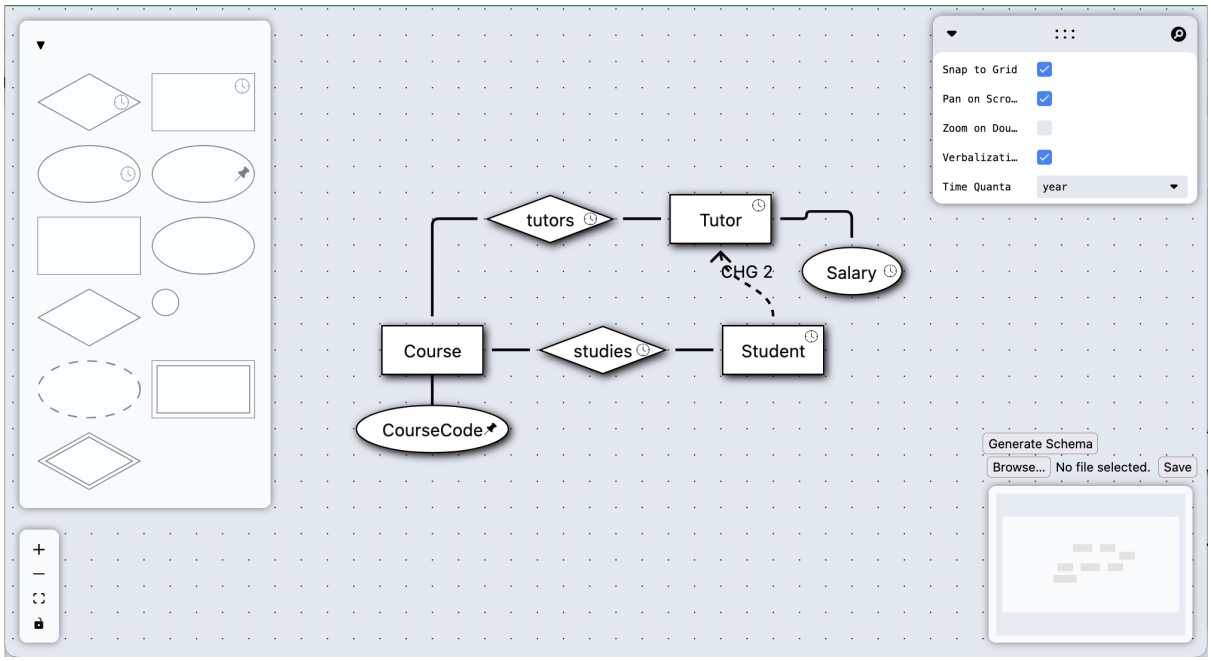


Figure 1: Screenshot of the TRENDy tool, with the TREND model of one of the evaluation tasks.

The remainder of this demo paper presents the front-end with the graphical modelling of logic-based temporal models in Section 2 and the transformation into SQL in Section 3. We then conclude, look ahead, and outline what can be expected at the demo in Section 4.

2. TRENDy's graphical editor as front-end

TRENDy is built using the Model-View-Controller (MVC) design pattern, ensuring a robust and scalable framework. The Model serialisation is implemented as a JSON object, maintaining separate lists for nodes (the implementation of entities, attributes) and edges (the implementation of temporal constraints). Said lists contain all necessary information for on-screen rendering and integration with the SQL schema generation component (described in the next section). The View and Controller are managed by ReactFlow, a React library optimised for node-based UI design, enabling features like drag-and-drop, resizing, and an infinitely expanding canvas. The architecture allows easy long-term maintainability and adaptability of the codebase.

The interface with full layout is shown in Fig. 1. Key components include state management (therewith allowing the user to undo and redo changes on the model), basic syntax checking and explanation of errors, a verbalisation feature that provides natural language renderings of selected temporal constraints (see Fig. 2), and other features, such as snap-to-grid, panning, and saving.

TRENDy was evaluated through a user study involving 16 computer science students that had completed a database course including EER two years before. They were divided equally between TRENDy and the control tool Draw.io. The learning curve for TRENDy compared to the relatively well-known Draw.io was not factored into the experiment set-up. Participants were tasked with creating temporal models based on TREND's controlled natural language [10] that was generated from the pre-created models. Performance was measured in terms of task completion time and correctness (marking scheme as in Experiment 11 of [8]), and it was supplemented by the System Usability Scale (SUS) for usability assessment. A small incentive was provided to the participants as remuneration for their effort and time, being pizza for lunch. Ethics approval was obtained from the UCT Science Faculty Ethics Committee. See supplementary material on GitHub for experiment details.

The scatter plot in Fig. 3 shows the results for each participant. For Draw.io, the average mark was 12.63/21 (60.12%) and the average time to complete the tasks was 25:10 minutes; for TRENDy, they were

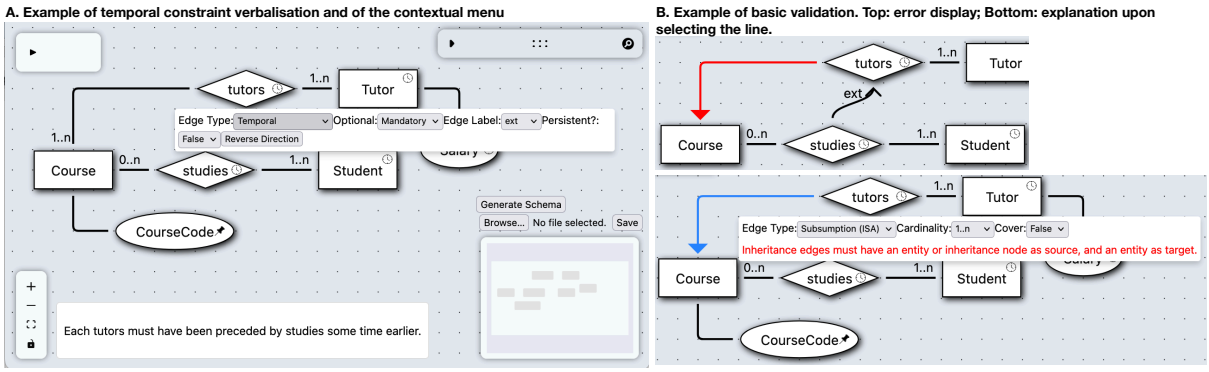


Figure 2: Screenshots of the TRENDy tool, displaying other features.

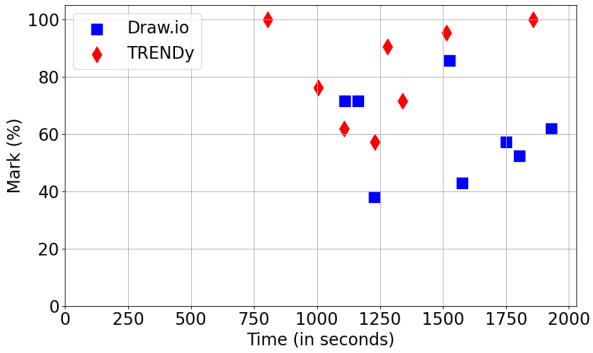


Figure 3: Scatter Plot showing Time and Marks for the task based evaluation.

17.13/21 (81.55%) and 21:07 minutes, respectively. Thus, TRENDy users were 16.01% faster with a 21.43% improvement in their model correctness, on average. The average SUS score was 43.13 ("poor") for Draw.io and 75.63 ("good") for TRENDy. The downside of familiarising oneself with a new tool of good quality was thus offset by faster, and better, modelling. Future interface evaluations might elucidate whether this can be attributed to the specific and guided options, cf. a flexible Draw.io palette, and/or additional model explanations in TRENDy.

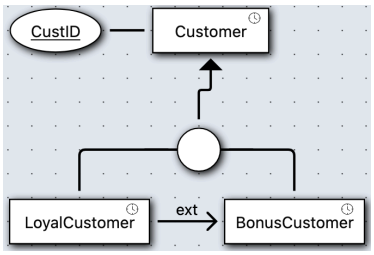
3. Converting a TREND model to SQL with TRENDy

This component of the tool converts TREND models into functional SQL databases. The SQL files are designed to work for MariaDB [11] and adhere to the SQL:2011 standard [12].

All standard features present in Extended Entity Relationship Diagrams are implemented, as well as the temporal logics and transition constraints of TREND. This was achieved by developing an algorithm that takes a serialised TREND model from the graphical editor and generates an SQL file containing the tables and triggers specified according to the model. The algorithm scales log linearly with input size converting a list of nodes and edges into data structures representing the finalised tables. Each of these data structures are then used to generate SQL strings which are appended to the output file.

For the temporal entities, it adds 'start' and 'end' columns for the valid time. The temporal constraints in TREND are converted into SQL triggers. The triggers for the temporal transition constraints prevent the insertion of any tuple that violates the formal semantics of TREND. Each temporal constraint in TREND maps to a trigger pattern, which is then completed with the relevant elements from the TREND model that is being converted. For instance, given two entities, A and B with identifiers, AID and BID respectively, the trigger to enforce an optional dynamic evolution in the past (chg, with a dotted arrow in the diagram) is the following:

A. Each bonus customer was already a loyal customer



B. TREND syntax

LoyalCustomer ISA_{C} Customer

BonusCustomer ISA_{C} Customer

ID(Customer) = CustID

$^{\text{mext}}$LoyalCustomer,BonusCustomer

C. Semantics of 'mandatory ext'

mext, *Mandatory* ext, *past*

$o \in \text{mext}^{\mathcal{I}(t)}_{C_1,C_2} \rightarrow (o \in {C_1}^{\mathcal{I}(t)} \rightarrow \exists t' < t . o \in \text{EXT}^{\mathcal{I}(t')}_{C_1,C_2})$

where C1 is the Loyal Customer and C2 the Bonus Customer and EXT is defined as:

EXT, *extension in the future, optional*

$o \in \text{EXT}^{\mathcal{I}(t)}_{C_1,C_2} \rightarrow (o \in {C_1}^{\mathcal{I}(t)} \wedge o \notin {C_2}^{\mathcal{I}(t)} \wedge o \in {C_2}^{\mathcal{I}(t+1)})$

D. Section of the SQL code generated

```sql
CREATE OR REPLACE TABLE Customer (
    CustID VARCHAR(30) NOT NULL,
    Customer_start DATE,
    Customer_end DATE,
    PERIOD FOR Customer_period(Customer_start, Customer_end),
    UNIQUE (CustID, Customer_period WITHOUT OVERLAPS),
    PRIMARY KEY (CustID, Customer_start, Customer_end)
);

CREATE OR REPLACE TABLE LoyalCustomer (
    CustID VARCHAR(30) NOT NULL,
    LoyalCustomer_start DATE,
    LoyalCustomer_end DATE,
    PERIOD FOR
      LoyalCustomer_period(LoyalCustomer_start, LoyalCustomer_end),
    UNIQUE (CustID, LoyalCustomer_period WITHOUT OVERLAPS),
    PRIMARY KEY (CustID, LoyalCustomer_start, LoyalCustomer_end),
    CONSTRAINT `LoyalCustomer_Customer_foreign_key`
        FOREIGN KEY (CustID) REFERENCES Customer(CustID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```sql
CREATE TRIGGER LoyalCustomer_ext_to_BonusCustomer
BEFORE INSERT ON BonusCustomer
FOR EACH ROW
BEGIN
    DECLARE initial_exists INT;
    SELECT COUNT(*)
    INTO initial_exists
    FROM LoyalCustomer
    WHERE NEW.CustID = CustID
        AND LoyalCustomer_start <= NEW.BonusCustomer_start;
    IF initial_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT =
          'BonusCustomer must extend from LoyalCustomer.';
    END IF;
END;
```

Figure 4: Small TREND model, its syntax, semantics of mext, and a section of the SQL code generated.

```sql
CREATE TRIGGER A_chg_to_B_1
BEFORE INSERT ON B
FOR EACH ROW
BEGIN
DECLARE initial_exists INT;
SELECT COUNT(*)
INTO initial_exists
FROM A
WHERE NEW.BID = AID
AND (
(NEW.B_end > A_start AND NEW.B_start <= A_start)
OR
(NEW.B_end >= A_end AND NEW.B_start < A_end)
OR
(NEW.B_start >= A_start AND NEW.B_end <= A_end)
);
IF initial_exists > 0 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'B cannot overlap with A.';
END IF;
END;
```

This trigger prevents entries being made in B's table that overlap with existing entries with the same identifier in A, enforcing the rule that "each object in B could have been in A, but is not in A anymore", by preventing an object from being in both A and B. An example with generated SQL code for the 'mandatory extension in the past' is shown in Fig. 4.

Thorough testing has shown that the tool can accurately generate schemas based on the user's models, and that these models obey TREND's semantics. This schema generation feature is expected to decrease the barrier to entry of developing temporal databases by reducing the time and costs associated with it, solving the problem of needing to manually implement all temporal features in the application logic.

4. Conclusions and Demo

This demo paper introduced the Web-based TRENDy tool, with which a user can develop temporal conceptual data models in TREND notation and convert it into a temporal relational database that also enforces the temporal constraints specified in the model. The small-scale evaluation with typical novice modellers showed the tool to outperform the baseline modelling tool.

This proof-of-concept constitutes enabling infrastructure, not only to simplify temporal modelling, but also for future experiments regarding uptake of temporal constraints, how a transformation-based OBDA pipeline may be designed, and to what extent that may overlap with requirements for temporal knowledge graphs. Either temporal SQL-to-RDF or a direct conversion to RDF with SHACL is another avenue of future work.

The demo will showcase all features and participants can try it themselves, be it from scratch or based on examples, such as those used in the user evaluation of the front-end. The machine will also run an instance of MariaDB so that the TRENDy-generated SQL can be executed to create the temporal database where the correct functioning of the temporal constraints can be verified.

Acknowledgments

We thank the participants in the evaluation.

References

[1] S. Batsakis, E. Petrakis, I. Tachmazidis, G. Antoniou, Temporal representation and reasoning in OWL 2, Semantic Web Journal 8 (2017) 981–1000.

[2] V. Milea, F. Frasincar, U. Kaymak, towl: a temporal web ontology language, in: IEEE Transactions on Systems, Man and Cybernetics. Part B, Cybernetics (IEEE T-SMC-Part B)„ volume 42, 2012, pp. 268–281.

[3] C. Lutz, F. Wolter, M. Zakharyaschev, Temporal description logics: A survey, in: Proc. of TIME'08, IEEE Computer Society Press, 2008.

[4] A. Artale, R. Kontchakov, A. Kovtunova, V. Ryzhikov, F. Wolter, M. Zakharyaschev, Ontology-mediated query answering over temporal data: A survey (invited talk), in: Proc. of TIME'17, volume 90 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 1:1–1:37. 2017, October 16-18, 2017, Mons, Belgium.

[5] Q. Gao, M. Lee, G. Dobbie, Z. Zeng, A semantic framework for designing temporal SQL databases, in: Proc of ER'18, volume 11157 of *LNCS*, Springer, 2018, pp. 382–396. Xi'an, China, October 22-25, 2018.

[6] P. R. Fillottrani, C. M. Keet, KnowID: An architecture for efficient knowledge-driven information and data access, Data Intelligence 2 (2020) 487–512.

[7] C. M. Keet, S. Berman, Determining the preferred representation of temporal constraints in conceptual models., in: H. Mayr, et al. (Eds.), Proc. of ER'17, volume 10650 of *LNCS*, Springer, 2017, pp. 437–450. 6-9 Nov 2017, Valencia, Spain.

[8] S. Berman, C. M. Keet, T. Shunmugam, The temporal conceptual data modelling language TREND, Technical report, 2024. URL: https://arxiv.org/abs/2408.09427. arXiv:2408.09427.

[9] A. Artale, E. Franconi, F. Wolter, M. Zakharyaschev, A temporal description logic for reasoning about conceptual schemas and queries, in: S. Flesca, S. Greco, N. Leone, G. Ianni (Eds.), Proc of JELIA'02, volume 2424 of *LNAI*, Springer Verlag, 2002, pp. 98–110.

[10] C. M. Keet, Natural language template selection for temporal constraints, in: Proc. of CREOL'17, part of JOWO'17, volume 2050 of *CEUR-WS*, 2017, p. 12. 21-23 September 2017, Bolzano, Italy.

[11] MariaDB, Mariadb server: the innovative open source database, 2024. URL: https://mariadb.org/.

[12] K. Kulkarni, J.-E. Michels, Temporal features in sql: 2011, ACM Sigmod Record 41 (2012) 34–43.