

THE NATIONAL UNIVERSITY  
of SINGAPORE



School of Computing  
Computing 1, 13 Computing Drive, Singapore 117417

**TRB3/18**

**A Semantic Framework for Designing Temporal  
SQL Databases**

*Qiao Gao, Mong Li Lee, Gillian Dobbie and Zhong Zeng*

*March 2018*

# Technical Report

## Foreword

*This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.*

Mohan KANKANHALLI  
Dean of School

# A Semantic Framework for Designing Temporal SQL Databases

Qiao Gao<sup>1</sup>, Mong Li Lee<sup>1</sup>, Gillian Dobbie<sup>2</sup>, Zhong Zeng<sup>3</sup>

<sup>1</sup>National University of Singapore, <sup>2</sup>University of Auckland

<sup>3</sup>Data Center Technology Lab, Huawei

{gaoqiao, leeml}@comp.nus.edu.sg;

g.dobbie@auckland.ac.nz; zengzhong4@huawei.com

**Abstract.** Many real world applications need to capture a mix of temporal and non-temporal entities, relationships and attributes. These concepts add complexity when designing database schemas and existing works are unable to capture the temporal semantics precisely. We propose a new framework for designing SQL databases that distinguishes between temporal and non-temporal concepts while also distinguishing between entities, relationships and attributes at every step. The framework first utilizes an Entity-Relationship (ER) diagram to capture the real world semantics. Temporal constructs in the ER diagram are then annotated. Finally we map the temporal ER diagram to a normal form database schema that reduces redundant data by separating current data from historical data. We also describe how data consistency is maintained during updates. Experiment results show that we can generate database schemas that support efficient access to both current and historical information, and enable better management of temporal data.

## 1 Introduction

Many organizations, especially in regulated industries such as finance and healthcare, need to manage and maintain data that changes over time. SQL:2011 [9] introduces temporal tables where a relational table can be associated with an explicit time period  $[Start, End)$  to restrict the valid times of its tuples. However, designing database schemas that capture both temporal and non-temporal data with entity, relationship and attribute semantics is complex, and may lead to data redundancy if the semantics are not carefully considered at every step of the design process.

A database that keeps the history of entities, relationships and their attributes will contain both current valid and historical data. Information about an entity/relationship and its temporal and non-temporal attributes may be stored over several tuples in multiple temporal and/or non-temporal tables. Temporal joins are needed to enforce constraints between the time periods of tuples from two or more temporal tables. For queries that primarily focus on current valid data, the volume of historical data will hamper the query evaluation process. Designing database schemas that reduce temporal joins and support efficient access to both current and historical information is crucial.

Employee						
Eid	Ename	Phone	Salary	Start	End	
$t_{11}$	e01	Alice	9000001	5,000	2000-01-01	2010-01-01
$t_{12}$	e01	Alice	9000001	7,000	2010-01-01	2012-01-01
$t_{13}$	e01	Alice	9000001	8,000	2012-01-01	9999-12-31
$t_{14}$	e02	Bob	9000002	6,000	2008-01-01	2012-01-01
$t_{15}$	e02	Bob	9000002	7,000	2012-01-01	2017-05-01

EmployeeHobby				
Eid	Hobby	Start	End	
$t_{21}$	e01	Photography	1998-01-01	2005-01-01
$t_{22}$	e01	Running	2003-01-01	9999-12-31
$t_{23}$	e01	Swimming	2000-01-01	9999-12-31
$t_{24}$	e02	Basketball	2005-01-01	2017-05-01
$t_{25}$	e02	Running	2014-01-01	2017-05-01

Department				
Did	Dname	Start	End	
$t_{31}$	d01	Research and Development	1995-10-01	9999-12-31
$t_{32}$	d01	Product	1990-01-01	9999-12-31

Project					
Jid	Title	Budget	Start	End	
$t_{41}$	j01	Social Network	80,000	2007-03-01	2009-01-01
$t_{42}$	j02	Data Mining	50,000	2008-05-20	2010-08-01
$t_{43}$	j03	Keyword Search	70,000	2012-10-01	2014-01-01
$t_{44}$	j04	Deep Learning	80,000	2015-01-01	9999-12-31

EmpDep					
Eid	Did	Position	Start	End	
$t_{41}$	e01	d02	Engineer	2000-01-01	2005-01-01
$t_{42}$	e01	d01	Engineer	2005-01-01	2012-01-01
$t_{43}$	e01	d01	Senior Engineer	2010-01-01	2012-01-01
$t_{44}$	e01	d01	Manager	2012-01-01	9999-12-31
$t_{45}$	e02	d01	Engineer	2008-01-01	2012-01-01
$t_{46}$	e02	d02	Technical Support	2012-01-01	2013-01-01
$t_{47}$	e02	d01	Engineer	2013-01-01	2017-05-01

EmpProj				
Eid	Pid	Start	End	
$t_{51}$	e01	j01	2007-03-01	2009-01-01
$t_{52}$	e01	j02	2008-05-20	2010-08-01
$t_{53}$	e01	j03	2012-10-01	2014-01-01
$t_{54}$	e01	j04	2015-01-01	9999-12-31
$t_{55}$	e02	j03	2013-01-01	2014-01-01

Fig. 1. An example temporal company database with problematic design

Fig. 1 shows an example company database whose data are stored using the temporal tables in SQL:2011. The schema of the database was designed using a standard technique of deriving tables from an ER diagram and adding attributes *Start* and *End* to each table to indicate the validity of the data values.

**Temporal semantics.** The complexity of mixing temporal and non-temporal data may lead to difficulty in enforcing the intended temporal semantics when updates occur. Consider the temporal table *Employee* which has a non-temporal attribute *Phone* whose value may change over time but only the current value is of interest and captured, and a temporal attribute *Salary* whose changes over time are tracked. Suppose an employee Alice wants to update her phone number to *90000011* on 2017-03-01. There are two possible ways to do the update, both of which are problematic:

- Set the valid end time of the tuple  $t_{13}$  to 2017-03-01, and insert a new tuple  $\langle e01, Alice, 8000, 90000011, 2017-03-01, 9999-12-31 \rangle$  to the table *Employee*, where the date “9999-12-31” means *now*. However, this violates the user requirement to keep only the latest phone number, that is, *Phone* is non-temporal, because tuples  $t_{11}$  and  $t_{12}$  still maintain the previous phone number of Alice.
- Set the value of attribute *Phone* in the tuples  $t_{11}, t_{12}, t_{13}$  to Alice’s new phone number. In this case, the tuple  $t_{11}$  becomes  $\langle e01, Alice, 90000011, 5000, 2000-01-01, 2010-01-01 \rangle$  which does not provide the correct valid time for Alice’s phone number.

**Data redundancy.** Having a schema where a table has both temporal and non-temporal attributes may lead to data redundancy when a temporal attribute in the table is updated. For instance, if *Alice* ( $e01$ ) has a salary raise on 2012-01-01 from \$7000 to \$8000, a new tuple  $t_{13}$  will be inserted into the *Employee*

table, and the end date of the tuple  $t_{12}$  is set to 2012-01-01. Note that *Alice*'s non-temporal attributes *Ename* and *Phone* are replicated, since we store them together with the temporal attribute *Salary* in one relation. Further, the temporal table *EmpDep* captures the temporal relationship between employees and departments. This table also has a temporal attribute *Position*, and any change in the job position of an employee will lead to replication of data.

**Costly Query Evaluation.** Processing a routine query involving temporal tables can become complex and costly. If a user wants to find the department that *Alice* currently works in, we need to carry out temporal joins of the tables *Employee*, *EmpDep* and *Department* to retrieve the current records of *Alice*. The historical records in each table participating in the temporal join will slow down the query evaluation process.

To address the above issues, we propose a framework that utilizes a high-level conceptual model, such as the Entity-Relationship model, to facilitate the database design process. We first construct a normal form ER diagram without considering the temporal aspects [11]. Then we annotate the entity types, relationship types and attributes based on the temporal aspects of the database. Finally, we map the temporal ER diagram to a set of normal form relations. We automatically generate two sets of normal form relations to increase the speed of queries, one for the current database state, and another for the historical database state. The key contributions of the mapping algorithm include

- a. Mapping each temporal and all non-temporal attributes to separate tables, thus reducing data redundancy;
- b. Associating time periods with each temporal concept such as entity, relationship and attribute in the same relation instead of the tuple, further reducing data duplication;
- c. Separating the current and historical data of entities and relationships, thus increasing the speed of queries.

With the separation of current and historical data, we examine how data consistency is maintained when there are updates. Experimental results show that our framework leads to temporal database schemas that reduce data redundancy, and supports efficient querying of both current and historical information.

## 2 Temporal ER Diagram

An ER diagram models real world entities that interact with each other via relationships, and their attributes. The works in [5, 4, 7] have proposed temporal ER diagrams to capture the temporal semantics in the real world. Here, we describe a temporal ER diagram ( $ERD^T$ ) that contains the essential constructs required for our mapping algorithms. We also analyze the semantics when we have temporal constructs, e.g., temporal entities that participate in non-temporal relationships, or non-temporal entities that participate in temporal relationships.

Capturing both real world semantics and temporal aspects of entities and relationships typically makes an ER diagram complicated and hard to design. As

such, we propose to first use an ER diagram to capture the entities, relationships and attributes of a database application, and then annotate these constructs with a superscript  $T$  to indicate that they are temporal. We have 3 types of temporal constructs depending on the changes that an application wants to keep track of.

**Temporal entity type.** A temporal entity type indicates that all the entities of this type are temporal. Each entity is associated with some implicit time period values  $[Start, End)$ . The entity is valid within these time periods.

**Temporal relationship type.** A temporal relationship type indicates that all the relationships of this type are temporal. A relationship may be associated with more than one time period values.

**Temporal attribute.** A temporal attribute indicates that the value of this attribute of an entity or relationship may change over time periods and the database keeps track of the changes.

An  $ERD^T$  can have various combinations of non-temporal and temporal constructs. For instance, a temporal entity/relationship type can have non-temporal attributes, and a non-temporal entity/relationship can have temporal attributes. Similarly, not all the participating entity types of a temporal relationship need to be temporal, and some participating entity types of a non-temporal relationship can be temporal. Note that an  $ERD^T$  becomes a traditional ER diagram if all its entity types, relationship types and attributes are non-temporal.

Fig. 2 shows the temporal ER diagram of a company database. *Department* is a *non-temporal entity type*. *Employee* is a non-temporal entity type with *temporal attributes* *Salary* and *Hobby*. *Project* is a *temporal entity type*. *EmpDep* is a temporal relationship type with a temporal attribute *Position*.

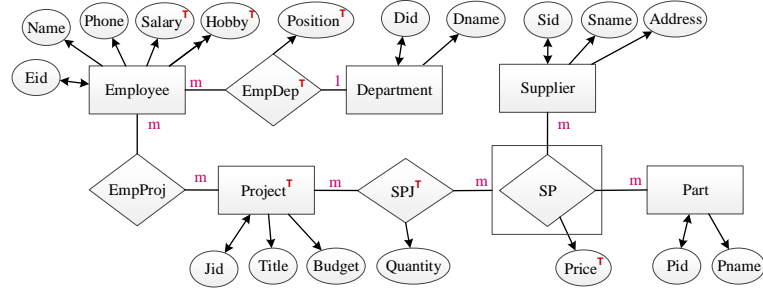


Fig. 2. A temporal ER diagram for a company database

### 3 Mapping Algorithms

In this section, we present our mapping algorithms to generate the schema for a temporal database from a temporal ER diagram. The schema separates the current and historical data to facilitate the processing of queries. Depending on the workload of the application, the schema can be a set of normal form relations or a set of nested relations. The latter facilitates the frequent retrieval of all the information pertaining to some entity/relationship.

### 3.1 Current Database Schema

We can store the current state of the database in a set of normal form relations. Since the temporal ER diagram models time period attributes (*Start* and *End*) implicitly, these attributes will be added to the database schema. Details of the mapping is given in Algorithm 1. The key steps of the mapping are:

- a. All the non-temporal or temporal single-valued attributes of a non-temporal or temporal entity/relationship type are mapped to one relation, and each non-temporal/temporal multivalued attribute together with the identifier of its entity/relationship type is mapped to a separate relation.
- b. For each temporal entity/relationship type in  $ERD^T$ , we add an attribute  $R\_Start$  to its corresponding current relation  $R\_curr$ . This  $R\_Start$  attribute indicates the start time of the temporal entities/relationships.
- c. For each temporal attribute  $A^T$ , we add an attribute  $A^T\_Start$  to the current relation  $R\_curr$  containing this temporal attribute  $A^T$  to indicate the start time of the attribute values of  $A^T$ .

Note that all the single valued attributes (temporal or non-temporal) are put in one relation and a start time for each of the temporal single valued attributes and a start time for the entity/relationship types is added if the entity/relationship type is temporal. This removes the issue of data redundancy, since the current relation has only one current value for each single-valued temporal attribute. Since our  $ERD^T$  is in normal form and the current state database is a snapshot database, the relations obtained are in 3NF or 4NF [11].

**Key of a current relation.** The identifier of a temporal/non-temporal entity type becomes the key of the corresponding current relation for its single valued attributes. The current relation corresponding to a temporal/non-temporal relationship type contains the identifiers of its participating entity types. For the current relation corresponding to a temporal/non-temporal multivalued attribute, the identifier of its entity/relationship type together with this multivalued attribute form the key. Note that the *Start* time of a temporal entity/relationship/attribute is not part of the key.

*Example 1.* [Current Data] Consider the  $ERD^T$  in Fig. 2. The following are some normal form relations generated for the current state of the database. The key of a relation is underlined.

1. Non-temporal entity type **Employee** with temporal attributes **Salary** and **Hobby**:  
 $Employee\_curr(\underline{Eid}, Name, Phone, Salary, Salary\_Start)$   
 $EmployeeHobby\_curr(\underline{Eid}, Hobby, Hobby\_Start)$
2. Temporal entity type **Project** with non-temporal attributes **Title** and **Budget**:  
 $Project\_curr(\underline{Jid}, Title, Budget, Project\_Start)$
3. Non-temporal relationship type **SP** with temporal attribute **Price**:  
 $SP\_curr(\underline{Sid}, \underline{Pid}, Price, Price\_Start)$
4. Temporal relationship type **SPJ** with non-temporal attribute **Quantity**:  
 $SPJ\_curr(\underline{Sid}, \underline{Pid}, Jid, Quantity, SPJ\_Start)$
5. Temporal relationship type **EmpDep** with temporal attribute **Position**:  
 $EmpDep\_curr(\underline{Eid}, \underline{Did}, Position, Position\_Start, EmpDep\_Start)$

**Algorithm 1:** Generate Current Database Schema

---

**Input:** normal form temporal ER diagram  $D$   
**Output:** set of normal form current relations  $\mathcal{R}$

- 1  $\mathcal{R} = \emptyset$ ;
- 2 **foreach** entity type  $E \in D$  with identifier  $K$  and set of single-valued attributes  $S$  **do**
- 3     Let  $S$  be the set of single-valued attributes;
- 4     Create a relation  $\mathbf{R}$  with attributes  $K \cup S$ , and  $K$  is the key of  $\mathbf{R}$ ;
- 5     **if**  $E$  is a temporal entity type **then**
- 6         | Add attribute  $R\_Start$  to  $\mathbf{R}$  to indicate the start time of  $E$ ;
- 7     **foreach** temporal single-valued attribute  $A^T \in S$  **do**
- 8         | Add attribute  $A^T\_Start$  to  $\mathbf{R}$  to indicate the start time of  $A^T$ ;
- 9      $\mathcal{R} = \mathcal{R} \cup \{\mathbf{R}\}$ ;
- 10    **foreach** non-temporal multivalued attribute  $A$  of  $E$  **do**
- 11         Create a relation  $\mathbf{R}'(K, A)$  with key  $A$  if  $A$  is a 1:m attribute, else  $\mathbf{R}'$  is an all-key relation;
- 12         **if**  $A$  is a 1:m attribute **then**
- 13             |  $A$  is the key of  $\mathbf{R}'$ ;
- 14         **else**
- 15             |  $\mathbf{R}'$  is a all-key relation;
- 16          $\mathcal{R} = \mathcal{R} \cup \{\mathbf{R}'\}$ ;
- 17    **foreach** temporal multivalued attribute  $A^T$  of  $E$  **do**
- 18         Create a relation  $\mathbf{R}_t(K, A^T)$  with key  $A^T$  if  $A^T$  is a 1:m attribute, else  $A^T$  and  $K$  form the key of  $\mathbf{R}_t$ ;
- 19         Add attribute  $A^T\_Start$  to  $\mathbf{R}_t$  to indicate the start time of  $A^T$ ;
- 20         **if**  $A^T$  is a 1:m attribute **then**
- 21             |  $A^T$  is the key of  $\mathbf{R}_t$ ;
- 22         **else**
- 23             |  $A^T$  and  $K$  form the key;
- 24          $\mathcal{R} = \mathcal{R} \cup \{\mathbf{R}_t\}$ ;
- 25 **foreach** relationship type  $R \in D$  with identifier  $K$  and set of single-valued attributes  $S$  **do**
- 26     Let  $P$  be the set of all the identifiers of the participating entity types;
- 27     Let  $S$  be the set of single-valued attributes of  $R$ ;
- 28     Create a relation  $\mathbf{R}$  with attributes  $P \cup S$ , and  $K$  is the key of  $\mathbf{R}$ ;
- 29     **if**  $R$  is a temporal relationship type **then**
- 30         | Add attribute  $R\_Start$  to  $\mathbf{R}$  to indicate the start time of  $R$ ;
- 31     **foreach** temporal single-valued attribute  $A^T \in S$  **do**
- 32         | Add attributes  $A^T\_Start$  to  $\mathbf{R}$  to indicate the start time of  $A^T$ ;
- 33      $\mathcal{R} = \mathcal{R} \cup \{\mathbf{R}\}$ ;
- 34    **foreach** non-temporal multivalued attribute  $A$  of  $R$  **do**
- 35         Create a relation  $\mathbf{R}'(K, A)$  with key  $A$  if  $A$  is a 1:m attribute, else  $\mathbf{R}'$  is an all-key relation;
- 36         **if**  $A$  is a 1:m attribute **then**
- 37             |  $A$  is the key of  $\mathbf{R}'$ ;
- 38         **else**
- 39             |  $\mathbf{R}'$  is a all-key relation;
- 40          $\mathcal{R} = \mathcal{R} \cup \{\mathbf{R}'\}$ ;
- 41    **foreach** temporal multivalued attribute  $A^T$  of  $R$  **do**
- 42         Create a relation  $\mathbf{R}_t(K, A^T)$  with key  $A^T$  if  $A^T$  is a 1:m attribute, else  $A^T$  and  $K$  form the key of  $\mathbf{R}_t$ ;
- 43         Add attribute  $A^T\_Start$  to  $\mathbf{R}_t$  to indicate the start time of  $A^T$ ;
- 44         **if**  $A^T$  is a 1:m attribute **then**
- 45             |  $A^T$  is the key of  $\mathbf{R}_t$ ;
- 46         **else**
- 47             |  $A^T$  and  $K$  form the key;
- 48          $\mathcal{R} = \mathcal{R} \cup \{\mathbf{R}_t\}$ ;
- 49 **return**  $\mathcal{R}$ ;

---

Note that we can have more than one start time in a relation, e.g. in relation *EmpDep\_curr*, the start time for relation *EmpDep\_Start* records the existence of relationship *EmpDep*, while attribute *Position\_Start* records the start time of temporal attribute *Position*.  $\square$

### 3.2 Historical Database Schema

Our mapping algorithm separates the current and historical data by generating another set of relations for the historical data. Generating the normal form relations for the historical database is similar to that for the current database. The main differences are:

- a. For each temporal entity/relationship type in  $ERD^T$ , its corresponding historical relation *R\_hist* contains both *R\_Start* and *R\_End* attributes indicating the start and end time of the temporal entities/relationships.
- b. For each temporal single valued or multivalued attribute *A*, we generate a separate historical relation which contains the primary key of its original current relation together with a time period, i.e. *A\_Start* and *A\_End* attributes.
- c. Each non-temporal multivalued attribute *A* of a temporal entity/relationship type is associated with *R\_Start* and *R\_End* attributes to indicate the set of attribute values for each time period of the corresponding relationship *R*.
- d. The non-temporal attributes of non-temporal entity/relationship type is not stored in any historical relations.

Note that we separate temporal attributes and non-temporal attributes in the historical relations. Further, we create one historical table for each temporal attribute and each temporal entity/relationship type.

**Key of historical relation.** The start time *R\_Start* for a temporal entity or relationship type *R* and the start time *A\_Start* for an attribute *A* are part of the key in the corresponding historical relations.

*Example 2.* [Historical Data] Consider the  $ERD^T$  in Fig. 2. The relations to store the historical information are as follows:

1. Non-temporal entity type **Employee** with temporal attributes:  
*EmployeeSalary\_hist*(*Eid*, *Salary\_Start*, *Salary\_End*, *Salary*)  
*EmployeeHobby\_hist*(*Eid*, *Hobby*, *Hobby\_Start*, *Hobby\_End*)
2. Temporal entity type **Project**:  
*Project\_hist*(*Jid*, *Project\_Start*, *Project\_End*, *Title*, *Budget*)
3. Non-temporal relationship type **SP** with temporal attribute **Price**:  
*SP\_hist*(*Sid*, *Pid*, *Price\_Start*, *Price\_End*, *Price*)
4. Temporal relationship type **SPJ**:  
*SPJ\_hist*(*Sid*, *Pid*, *Jid*, *SPJ\_Start*, *SPJ\_End*, *Quantity*)
5. Temporal relationship type **EmpDep** with temporal attribute **Position**:  
*EmpDep\_hist*(*Eid*, *EmpDep\_Start*, *EmpDep\_End*, *Did*)  
*EmpDepPosition\_hist*(*Eid*, *Position\_Start*, *Position\_End*, *Did*, *Position*)  $\square$

### 3.3 Schema with Nested Relations

In order to minimize the cost of temporal joins, we propose to store all the attributes of an entity/relationship type in one nested relation. Since the  $ERD^T$  is in normal form, we can generate a set of normal form nested relations [12] for the  $ERD^T$  including the *Start* and *End* attributes.

**Current Nested Relations.** The only difference between the nested relations for current data and the normal form relations lies in the mapping of multivalued attributes to a first-level repeating group inside the relation of its corresponding entity/relationship type. This removes the need to join the relations for the entity/relationship and the multivalued attribute.

*Example 3.* [Current Data in Nested Relations] Consider the  $ERD^T$  in Fig. 2. *Nested* relations generated for the current state of the database include

1. Non-temporal entity type **Employee** with temporal attributes **Salary** and **Hobby**:  
 $Employee\_currN(\underline{Eid}, Name, Phone, Salary, Salary\_Start, (\underline{Hobby}, Hobby\_Start)^*)$
2. Temporal entity type **Project** with non-temporal attributes **Title** and **Budget**:  
 $Project\_currN(\underline{Jid}, Title, Budget, Project\_Start)$
3. Non-temporal relationship type **SP** with temporal attribute **Price**:  
 $SP\_currN(\underline{Sid}, \underline{Pid}, Price, Price\_Start)$
4. Temporal relationship type **SPJ** with non-temporal attribute **Quantity**:  
 $SPJ\_currN(\underline{Sid}, \underline{Pid}, \underline{Jid}, Quantity, SPJ\_Start)$
5. Temporal relationship type **Join** with temporal attribute **Position**:  
 $EmpDep\_currN(\underline{Eid}, \underline{Did}, Position, Position\_Start, EmpDep\_Start)$

Note that the multivalued temporal attribute *Hobby* is stored in a first-level repeating group in relation *Employee\_curr*. □

**Historical Nested Relations.** The intuition behind historical nested relations is to put each single-valued or multivalued temporal attribute with its start and end time attributes into a repeating group, since a temporal attribute becomes a multivalued attribute in historical relation.

For each temporal entity type, all the non-temporal single-valued attributes together with the attributes *R\_Start* and *R\_End* of the time period for the entity form the first-level repeating group of the nested relation *R*. This time period constrains the temporal entity as well as all its attributes. Each non-temporal multivalued attribute forms a second-level repeating group since it is constrained by the time period of the temporal entity. Each single-valued or multivalued temporal attribute *A* together with their time period *A\_Start* and *A\_End* also form a second-level repeating group.

For each temporal relationship type, all its non-temporal single-valued attributes together with *R\_Start* and *R\_End* attributes form a first-level repeating group for the relation *R*. This time period constrains the temporal relationship as well as all its attributes. Each multivalued non-temporal attribute forms a second-level repeating group since it is constrained by the time period of the temporal relationship. Each single-valued or multivalued temporal attribute together with their start and end times also forms a second-level repeating group.

Note that if an entity/relationship type is non-temporal but has some temporal attributes, we generate the nested relation that contains only the temporal attributes as repeating groups. Details of the mapping algorithm is given in Algorithm 2.

---

**Algorithm 2:** Generate Nested Relations for Historical Database Schema
 

---

**Input:** normal form temporal ER diagram  $D$   
**Output:** set of nested relations  $\mathcal{R}_N$

```

1  $\mathcal{R}_N = \emptyset$ ;
2 foreach entity/relationship type  $E \in D$  with identifier  $K$  and set of
   non-temporal single-valued attributes  $S$  do
3   if  $E$  is a temporal entity/relationship type or a non-temporal
   entity/relationship type with some temporal attributes then
4     if  $E$  is an entity type then
5       | Create a relation  $\mathbf{R}$  with attributes  $K$  and  $K$  is the key of  $\mathbf{R}$ ;
6     if  $E$  is an relationship type then
7       | Let  $P$  be the set of all the identifiers of the participating entity
       | types of  $E$ ;
8       | Create a relation  $\mathbf{R}$  with attributes  $P$ , and  $K$  is the key of  $\mathbf{R}$ ;
9     if  $E$  is a temporal entity/relationship type then
10      | Create a first-level repeating group  $G_1(S, R\_Start, R\_End)$  in  $\mathbf{R}$ 
      | with  $R\_Start$  as the key of  $G_1$ ;
11      | foreach non-temporal multivalued attribute  $A$  of  $E$  do
12      | | Create a second-level repeating group  $G_2(A)$  in  $G_1$  with  $A$  as
      | | the key of  $G_2$ ;
13      | foreach temporal attribute  $A^T$  of  $E$  do
14      | | if  $R$  is a temporal entity/relationship type then
15      | | | Create a second-level repeating group
      | | |  $G_2(A^T, A^T\_Start, A^T\_End)$  in  $G_1$ ;
16      | | else
17      | | | Create a first-level repeating group  $G_1(A^T, A^T\_Start, A^T\_End)$ 
      | | | in  $\mathbf{R}$ ;
18      | | if  $A^T$  is a 1:1 or m:1 attribute then
19      | | |  $A^T\_Start$  is the key of  $G$ ;
20      | | else
21      | | |  $A^T\_Start$  and  $A^T$  form the key of  $G$ ;
22      |  $\mathcal{R}_N = \mathcal{R}_N \cup \{\mathbf{R}\}$ ;
23 return  $\mathcal{R}_N$ ;

```

---

**Key for historical nested relation.** The start time  $R\_Start$  for a temporal entity or relationship type  $R$  and the start time  $A\_Start$  for an attribute  $A$  is a part of the key for its corresponding historical relation or repeating group.

*Example 4.* [Historical Data in Nested Relations] Consider the  $ERD^T$  in Fig. 2. The nested relations to store the historical information of temporal entities, relationships and attributes are generated as follows. “(\*)” denotes a repeating group, and underlined attributes in a group form the key of this group.

1. Non-temporal entity type **Employee** with two temporal attributes:  
 $Employee\_histN(\underline{Eid}, (Salary, \underline{Salary\_Start}, Salary\_End)^*,$   
 $(\underline{Hobby}, \underline{Hobby\_Start}, Hobby\_End)^*)$
2. Temporal entity type **Project**:  
 $Project\_histN(\underline{Jid}, (Title, Budget, \underline{Project\_Start}, Project\_End)^*)$
3. Non-temporal relationship type **SP** with temporal attribute **Price**:  
 $SP\_histN(\underline{Sid}, \underline{Pid}, (Price, \underline{Price\_Start}, Price\_End)^*)$
4. Temporal relationship type **SPJ**:  
 $SPJ\_histN(\underline{Sid}, \underline{Pid}, \underline{Jid}, (Quantity, \underline{SPJ\_Start}, SPJ\_End)^*)$
5. Temporal relationship type **EmpDep** with temporal attribute **Position**:  
 $EmpDep\_histN(\underline{Eid}, \underline{Did}, ((Position, \underline{Position\_Start}, Position\_End)^*,$   
 $\underline{EmpDep\_Start}, EmpDep\_End)^*)$

Note that the historical relation for the non-temporal entity type **Employee** does not contain non-temporal attributes (e.g. *Name*). This can be obtained by joining the historical relation with the corresponding current relation in Example 3.  $\square$

## 4 Maintaining Data Consistency

Since we separate the current and historical data, we need to ensure consistency when updates occur. Fig. 3 shows the schema generated from the temporal ER diagram in Fig. 2. We focus on updates to the current data since historical data are typically not updated but archived for subsequent analysis. Update operations on the current data include modifying the values of temporal and non-temporal attributes, and inserting or deleting tuples. These tuples may correspond to multivalued attributes or some entity/relationship.

**Modify Attribute Values.** If a non-temporal attribute such as an employee’s phone number is modified, we simply replace the old attribute value in the current relation with the new value since the database does not keep the old values of non-temporal attributes. However, if a temporal attribute such as an employee’s salary is changed, we need to insert a corresponding tuple containing the old attribute value with a valid end time to the historical relation. The start date for the new salary value in the current relation is changed to reflect the valid time of the change.

Fig. 4 shows the changes (in italics) in the current and historical **Employee** relations after updating the phone number and salary of *Alice* to 90000011 on 2017-03-01 and \$10,000 on 2017-06-01 respectively.

**Insert/Delete Tuples.** Inserting new tuples to the current data does not affect the historical data, and will not lead to any inconsistency. However, deleting a tuple  $t$  from a current relation  $R\_curr$  may require the insertion of a corresponding tuple to the historical relation with valid end time. We have 3 cases:

1.  $R\_curr$  corresponds to a multivalued attribute of some entity/relationship. If the attribute is temporal, we insert a corresponding tuple  $t'$  to  $R\_hist$  with a valid end time, and delete  $t$  from  $R\_curr$ . Otherwise, we simply delete  $t$  from  $R\_curr$ . For example, if we delete tuple  $\langle e01, \textit{Swimming}, 2000-01-01 \rangle$

Employee_curr					Project_curr				EmpProj_curr	
Eid	Ename	Phone	Salary	Salary_Start	Jid	Title	Budget	Project_Start	Eid	Pid
e01	Alice	90000001	8,000	2012-01-01	j04	Deep Learning	80,000	2015-01-01	e01	j04
e03	Smith	90000003	7,000	2014-01-01	j05	Temporal Database	60,000	2016-01-01	e03	j04
e04	Brown	90000004	6,000	2015-01-01					e03	j05
									e04	j05

EmployeeHobby_curr			EmpDep_curr					Department_curr	
Eid	Hobby	Hobby_Start	Eid	Did	Position	Position_Start	EmpDep_Start	Did	Dname
e01	Running	1998-01-01	e01	d01	Manager	2012-01-01	2000-01-01	d01	Research and Development
e01	Swimming	2000-01-01	e03	d01	Engineer	2005-01-01	2012-01-01	d02	Product
e03	Running	2005-01-01	e04	d02	Technical Support	2010-01-01	2013-01-01		
e04	Reading	2007-01-01							

EmployeeHobby_hist				EmployeeSalary_hist				EmpDep_hist			
Eid	Hobby	Hobby_Start	Hobby_End	Eid	Salary	Salary_Start	Salary_End	Eid	Did	EmpDep	EmpDep

Employee_curr					EmployeeSalary_hist			
Eid	Ename	Phone	Salary	Salary_Start	Eid	Salary	Salary_Start	Salary_End
e01	Alice	90000011	10,000	2017-06-01	e01	5,000	2000-01-01	2010-01-01
e03	Smith	90000003	7,000	2014-01-01	e01	7,000	2010-01-01	2012-01-01
e04	Brown	90000004	6,000	2015-01-01	e03	6,000	2012-01-01	2014-01-01
					e04	5,000	2013-01-10	2015-01-01
					e01	8000	2012-01-01	2017-06-01

**Fig. 4.** Updated relations after Alice ( $e01$ ) changes her phone number and salary.

from *EmployeeHobby\_curr* in Fig. 3 with a valid end time 2017-08-01, we will insert a tuple  $\langle e01, \text{Swimming}, 2000-01-01, 2017-08-01 \rangle$  with valid end time *2017-08-01* to the historical relation *EmployeeHobby\_hist*.

2.  $R_{curr}$  corresponds to an entity type  $t$  with identifier  $e$ .

Since the information of an entity may be stored across multiple relations, its deletion may trigger the deletion of tuples that correspond to its multivalued attributes and the relationships the entity participates in.

- Case 2(a) Entity type is non-temporal.

We delete all tuples with identifier  $e$  from current and historical relations, i.e., remove all information about this entity from the database.

- Case 2(b) Entity type is temporal.

We delete tuples with identifier  $e$  from the current relations, and insert corresponding tuples with valid end times in the historical relations.

For example, suppose *Alice* leaves the company and we delete the tuple  $\langle e01, \text{Alice}, 90000001, 8000, 2012-01-01 \rangle$  from *Employee\_curr* in Fig. 3. This will trigger the deletion of all the tuples with  $eid = e01$  from both current and historical relations. In contrast, suppose the entity project  $j04$  is completed in 2017-10-01 and we delete it from current relation *Project\_curr*. Since this is a temporal entity, the database will store its information in the historical relations. As such, we delete the tuples involving  $j04$  from the current relations and insert the corresponding tuples with valid end times in the historical relations. Note that if employee is a temporal entity type,

then we will need to create historical relations to store all the non-temporal single-valued and multivalued attributes of employees who are no longer with the company, i.e., they have been deleted from the current relations.

3.  $R_{curr}$  corresponds to a relationship type.  
If the relationship is temporal, then insert a corresponding tuple with valid end time into the historical relation  $R_{hist}$ . Otherwise, delete tuple  $t$  from  $R_{curr}$ . For example, if employee Smith ( $e03$ ) no longer works in the department  $d01$  on 2017-09-01, then we insert tuple  $\langle e03, d01, 2012-01-01, 2017-09-01 \rangle$  into the historical relation  $EmpDep_{hist}$  and a tuple  $\langle e03, d01, Engineer, 2005-01-01, 2017-09-01 \rangle$  is inserted into the historical relation  $EmpDepPosition_{hist}$ . The tuple  $\langle e03, d01, Engineer, 2005-01-01, 2012-01-01 \rangle$  is deleted from the current relation  $EmpDep_{curr}$ .

## 5 Experiments

We evaluate the performance of our approach for designing temporal databases. We use Oracle 12c Enterprise Edition hosted on Solaris 10 with 2.60 GHz CPU and 128 GB RAM. Oracle implements an object-relational model and supports multi-level nested tables as objects. We generate 3 schemas based on the Employee database<sup>1</sup> which contains employees who work for and manage departments in a company, and keeps track of the changes of the employees' salaries, job titles, departments worked for and managed.

- S1: This is the traditional schema which mixes current and historical data.

```
Dept(deptno, name)
Employee(empno, Employee_Start, Employee_End, birthdate, name, gender)
Emptitle(empno, title_Start, title_End, title)
Empsalary(empno, salary_Start, salary_End, salary)
Workfor(empno, Workfor_Start, Workfor_End, deptno)
Manage(empno, Manage_Start, Manage_End, deptno)
```

- S2: This is our normal form relations that separates current and historical data.

```
Dept_curr(deptno, name)
Employee_curr(empno, birthdate, name, gender, title, title_Start,
              salary, salary_Start, Employee_Start)
Workfor_curr(empno, deptno, Workfor_Start)
Manage_curr(empno, deptno, Manage_Start)
Employee_hist(empno, Employee_Start, Employee_End, birthdate, name, gender)
Emptitle_hist(empno, title_Start, title_End, title)
Empsalary_hist(empno, salary_Start, salary_End, salary)
Workfor_hist(empno, Workfor_Start, Workfor_End, deptno)
Manage_hist(empno, Manage_Start, Manage_End, deptno)
```

- S3: This is our normal form *nested* relations that stores current and historical data separately.

<sup>1</sup> <https://dev.mysql.com/doc/employee/en/>

*Dept\_currN*(deptno, name)  
*Employee\_currN*(empno, birthdate, name, gender, title, title\_Start, salary, salary\_Start, Employee\_Start)  
*Work\_for\_currN*(empno, deptno, Work\_for\_Start)  
*Manage\_currN*(empno, deptno, Manage\_Start)  
*Employee\_histN*(empno, (birthdate, name, gender, (title, title\_Start, title\_End)\*, (salary, salary\_Start, salary\_End)\*, Employee\_Start, Employee\_End)\*)  
*Work\_for\_histN*(empno, deptno, (Work\_for\_Start, Work\_for\_End)\*)  
*Manage\_histN*(empno, deptno, (Manage\_Start, Manage\_End)\*)

**Table 1.** Statistics of Datasets

Dataset	Size	Ratio of historical to current entity/relationship	# of time periods per temporal relationship/attribute
$D_1$	136 MB	$\sim 1 : 3.2$	$\sim 3.6 : 1$
$D_2$	583 MB	10 : 1	1 : 1
$D_3$	699 MB	1 : 1	10 : 1

**Table 2.** Queries for Employee database

$Q_1$	For the employees who are “Senior Engineer” now, find their start date
$Q_2$	Find the current number of employees in each department
$Q_3$	Find the maximum salary in the salary history of employees with <i>Eid</i> < 100000 who are currently working in the “Marketing” department
$Q_4$	Find the job titles of female employees with <i>Eid</i> < 1000 who have left the company
$Q_5$	Find the salary history of employee “Aris Iwayama”
$Q_6$	Find the last job title for all the employees who have left the company
$Q_7$	Find the number of departments for resigned employees born before 1960-01-01 had joined
$Q_8$	Find the employees who are “Senior Engineer” now but used to be a “Engineer”
$Q_9$	Find the employees who used to manage the “Marketing” department

We have 3 datasets for each schema: the original Employee dataset  $D_1$ , and two synthetically generated datasets  $D_2$  and  $D_3$ . We have 10 historical entities/relationships for each current entity/relationship in  $D_2$ , and each temporal relationship/attribute is associated with one time period. The purpose is to increase the scale of historical data in the whole dataset to evaluate our approach that separates the current and historical states of the database. In  $D_3$ , each current entity/relationship has 1 historical entity/relationship, and each temporal relationship/attribute is associated with 10 time periods. This is to increase the size of repeating groups in nested relations to evaluate our approach that stores historical data in nested relations. Table 1 shows the statistics of these datasets. Table 2 gives the descriptions of our 9 test queries, and Fig. 5 shows the CPU time of SQL execution for each query on the 3 datasets.

$Q_1$  and  $Q_2$  query the current data only. These queries run much faster on  $S_2$  and  $S_3$  compared to  $S_1$  on all 3 datasets. When the ratio of historical to current data is increased (as in  $D_2$ ), the gap in their runtime widens. Note that the query time for  $S_2$  and  $S_3$  is the same for these two queries, since they have the same set of current relations.

$Q_3$  to  $Q_5$  retrieve historical information of current/resigned employees.  $Q_3$  requires traditional join and an aggregation function on the temporal attribute.  $Q_4$  finds all historical records for a large set of entities, while  $Q_5$  finds the historical records for a few entities. All these queries need to retrieve the historical record

for each entity/relationship. *S3* gives the best performance on all 3 datasets as it reduces the joins between temporal attributes and their entities/relationships.

$Q_6$  to  $Q_7$  are queries on temporal relationships or constrain temporal attributes with some values. As the queries focus on either historical or current data, *S1* is slower than *S2*. *S3* is expensive since it is difficult to retrieve entities or relationships whose attributes satisfy some constraints from nested relations.

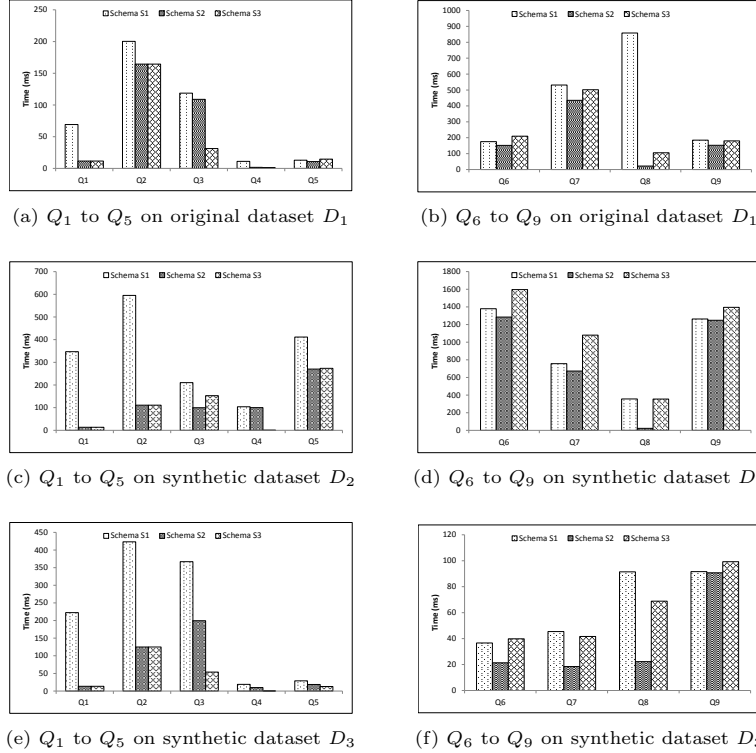


Fig. 5. CPU time of SQL execution for queries in Table 2

$Q_8$  contains a temporal join between current data and historical data. Since *S1* combines current and historical data, we need to join two **Emptitle** relations and check that the time period of “Engineer” is before that of “Senior Engineer”, which is time consuming. Separating current and historical data can avoid such temporal joins. *S3* does not perform as well as *S2* because this query constrains the value of the temporal attributes, which is slow for nested relations.  $Q_9$  retrieves both current and historical data. The runtime of this query is similar for all three schemas since we need to scan both current and historical data.

The results of our experiments demonstrate that separating the data into current and historical relations (for both flat and nested relations) improves the efficiency when queries are focused on either current or historical data. The normal form relations for storing historical data (*S2*) is a better design for temporal join between current and historical data, and is also good for querying historical

data with some constraints on temporal attributes. The nested relations which stores historical data (S3) shows good performance for queries that retrieve two or more historical records for entities/relationships.

## 6 Related Work

Research in conceptual modeling captures the temporal aspects of a database by introducing new temporal constructs to the standard ER diagram [8, 3, 1, 14, 13, 15], or extending existing constructs to include temporal semantics [2, 10]. Our  $ERD^T$  is a simplified temporal ER diagram that contains the essential constructs for our proposed mapping algorithm. Here, we use the symbol  $T$  to indicate that a construct (entity type, relationship type or attribute) is temporal and should be associated with a valid time when mapped to a temporal table.

There are two ways to map a temporal ER model to the temporal database schema. One way is to map the temporal ER model to the standard ER model first (such as adding explicit time period attributes) before translating the ER diagram to the database schema [13, 15]. The semantics of temporal entity, relationship and attribute are lost during the mapping since the time attributes, i.e., *Start* and *End*, are treated as regular attributes in the standard ER model.

Another way is to map the temporal conceptual model to a database schema directly [14, 10, 6]. [14] generates a set of 3NF database schema but its corresponding conceptual model does not support temporal attribute and attributes of relationship type. [10] adds an implicit valid time period for each generated relation, thus making all the attributes in the database temporal. This increases the data redundancy since it may have more than one temporal attributes in a relation. [6] designs a two-stage mapping process to map the temporal ER diagram to a surrogate-based relational model (RM/T model), which does not capture the semantics of temporal relationship type. All these methods do not separate the current data from historical data.

Commercial databases such as Microsoft SQL server and IBM DB2 10 separate current state and historical state relations for data involving transaction time, in which the historical state relation is a mirror relation of the current state table and used to track every modification in the current table. However, this approach cannot handle data involving valid time since it does not distinguish temporal and non-temporal data. Our proposed framework provides a principled approach to generate database schema that captures these temporal semantics and supports efficient access of data involving valid time.

## 7 Conclusion

The requirement to capture a mix of temporal and non-temporal entities, relationships and attributes adds complexity to the design of database schemas. We have proposed a semantic framework for designing temporal databases that precisely captures temporal semantics. Each step in the framework distinguishes temporal and non-temporal entities, relationships and attributes. Our mapping

algorithm generates current and historical schemas from the temporal ER diagram. Separating current and historical data accelerates the querying of current data, and providing the flexibility of mapping to a set of nested relations leads to fewer join operations. We also discussed how consistency between the current and historical data is maintained during updates especially when an entity/relationship is deleted. Our experiments show that our mapping algorithm generates database schemas that provide efficient access of both the current and historical information, and enables better management of temporal data.

## References

1. R. Elmasri, I. El-Assal, and V. Kouramajian. Semantics of temporal data in an extended ER model. In *ER*, 1990.
2. R. Elmasri and G. T. J. Wu. A temporal model and query language for er databases. In *IEEE ICDE*, 1990.
3. S. Ferg. Modelling the time dimension in an entity-relationship diagram. In *ER*, 1985.
4. H. Gregersen and C. Jensen. Conceptual modeling of time-varying information. Technical report, TimeCenter TR-35, 1998.
5. H. Gregersen and C. S. Jensen. Temporal entity-relationship models – a survey. *IEEE TKDE*, 1999.
6. H. Gregersen, L. Mark, and C. S. Jensen. Mapping temporal er diagrams to relational schemas. Technical report, Technical Report TimeCenter TR-39, 1998.
7. V. Khatri, S. Ram, R. T. Snodgrass, and P. Terenziani. Capturing telic/atelic temporal data semantics: Generalizing conventional conceptual models. *IEEE TKDE*, 2014.
8. M. R. Klopprogge. TERM: an approach to include time dimension in the entity-relationship model. In *ER*, 1981.
9. K. Kulkarni and J.-E. Michels. Temporal features in sql: 2011. *ACM Sigmod Record*, 2012.
10. V. S. Lai, J.-P. Kuilboer, and J. L. Guynes. Temporal databases: model design and commercialization prospects. In *ACM SIGMIS Database*, 1994.
11. T. W. Ling. A normal form for entity-relationship diagrams. In *ER*, 1985.
12. T. W. Ling and L. Yan. NF-NR: A practical normal form for nested relations. *Journal of Systems Integration*, 1994.
13. B. Tausovitch. Towards temporal extensions to the entity-relationship model. In *ER*, 1991.
14. C. Theodoulidis, P. Loucopoulos, and B. Wangler. A conceptual modelling formalism for temporal database applications. *Information Systems*, 1991.
15. E. Zimanyi, C. Parent, S. Spaccapietra, and A. Pirotte. Terc+: a temporal conceptual model. In *International Symposium Digital Media Information Base*, 1997.