A Semantic Framework for Designing Temporal SQL Databases

Qiao Gao[1(✉)], Mong Li Lee[1], Gillian Dobbie[2], and Zhong Zeng[3]

[1] National University of Singapore, Singapore, Singapore
{gaoqiao,leeml}@comp.nus.edu.sg
[2] University of Auckland, Auckland, New Zealand
g.dobbie@auckland.ac.nz
[3] Data Center Technology Lab, Huawei, Hangzhou, China
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 it is difficult 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.

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 [10] 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 incorrect temporal semantics and 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

© Springer Nature Switzerland AG 2018
J. C. Trujillo et al. (Eds.): ER 2018, LNCS 11157, pp. 382–396, 2018.
https://doi.org/10.1007/978-3-030-00847-5_27

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	90000001	5,000	2000-01-01	2010-01-01
t_{12}	e01	Alice	90000001	7,000	2010-01-01	2012-01-01
t_{13}	e01	Alice	90000001	8,000	2012-01-01	*9999-12-31*
t_{14}	e02	Bob	90000002	6,000	2008-01-01	2012-01-01
t_{15}	e02	Bob	90000002	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	Tittle	Budget	Start	End
t_{51}	j01	Social Network	80,000	2007-03-01	2009-01-01
t_{52}	j02	Data Mining	50,000	2008-05-20	2010-08-01
t_{53}	j03	Keyword Search	70,000	2012-10-01	2014-01-01
t_{54}	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_{61}	e01	j01	2007-03-01	2009-01-01
t_{62}	e01	j02	2008-05-20	2010-08-01
t_{63}	e01	j03	2012-10-01	2014-01-01
t_{64}	e01	j04	2015-01-01	*9999-12-31*
t_{65}	e02	j03	2013-01-01	2014-01-01

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

Figure 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:

a. Set the valid end time of the tuple t_{13} to 2017-03-01, and insert a new tuple <*e01, Alice, 8000, 90000011, 2017-03-01, 9999-12-31*> 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.

b. 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 <*e01, Alice, 90000011, 5000, 2000-01-01, 2010-01-01*> 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 (ER) model, to facilitate the database design process. We first construct a normal form ER diagram without considering the temporal aspects [12]. 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 [5,6,8] 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 combinations of non-temporal and temporal constructs, e.g., a temporal entity/relationship can have non-temporal attributes, and a non-temporal entity/relationship can have temporal attributes. 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. An ERD^T becomes a traditional ER diagram if all its entity types, relationship types and attributes are non-temporal. Figure 2 shows the temporal ER diagram

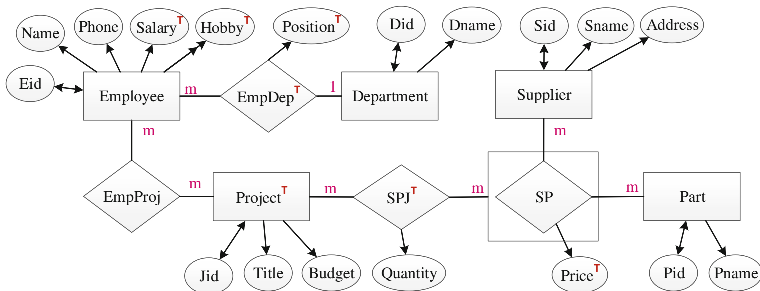


Fig. 2. A temporal ER diagram for a company database

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. Note that the temporal relationship type SPJ has an aggregate entity type SP.

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 entitiy/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 [4]. The key steps of the mapping are:

a. All non-temporal or temporal single-valued attributes of a non-temporal or temporal entity/relationship type are mapped to one relation.
b. Each non-temporal/temporal multivalued attribute together with the identifier of its entity/relationship type is mapped to a separate relation.
c. 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.
d. For each temporal attribute A, we add an attribute A_Start to the current relation R_curr containing this temporal attribute A to indicate the start time of the attribute values of A.

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 because 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 [12].

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]. Based on the ERD^T in Fig. 2, we generate some normal form relations for the current database schema.

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 Tittle and Budget:
   $Project_curr(\underline{Jid}, Title, Budget, Project_Start)$
3. Non-temporal relationship type SP with temporal attribute Price:
   $SP_curr(\underline{Sid, Pid}, Price, Price_Start)$
4. Temporal relationship type SPJ with non-temporal attribute Quantity:
   $SPJ_curr(\underline{Sid, Pid, Jid}, Quantity, SPJ_Start)$
5. Temporal relationship type EmpDep with temporal attribute Position:
   $EmpDep_curr(\underline{Eid}, Did, Position, Position_Start, EmpDep_Start)$

□

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*.

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 a temporal 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:
   $EmployeeSalary_hist(\underline{Eid, Salary_Start}, Salary_End, Salary)$
   $EmployeeHobby_hist(\underline{Eid, Hobby, Hobby_Start}, Hobby_End)$
2. Temporal entity type Project:
   $Project_hist(\underline{Jid, Project_Start}, Project_End, Title, Budget)$
3. Non-temporal relationship type SP with temporal attribute Price:
   $SP_hist(\underline{Sid, Pid, Price_Start}, Price_End, Price)$
4. Temporal relationship type SPJ:
   $SPJ_hist(\underline{Sid, Pid, Jid, SPJ_Start}, SPJ_End, Quantity)$
5. Temporal relationship type EmpDep with temporal attribute Position:
   $EmpDep_hist(\underline{Eid, EmpDep_Start}, EmpDep_End, Did)$
   $EmpDepPosition_hist(\underline{Eid, Position_Start}, Position_End, Did, Position)$

□

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 [13] for the ERD^T including the *Start* and *End* attributes.

Current Nested Relations. The 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:
   $SP_currN(\underline{Sid, Pid}, Price, Price_Start)$
4. Temporal relationship type SPJ with non-temporal attribute Quantity:
   $SPJ_currN(\underline{Sid, Pid, Jid}, Quantity, SPJ_Start)$
5. Temporal relationship type Join with temporal attribute Position:
   $EmpDep_currN(\underline{Eid}, Did, Position, Position_Start, EmpDep_Start)$

□

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/relationship type, all its non-temporal single-valued attributes together with attributes *R_Start* and *R_End* form the first-level repeating group of a nested relation R. The time period constrains the temporal entity/relationship 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/relationship. 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.

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 [4].

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 a temporal 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 repeating group form the key of this group.

1. Non-temporal entity type Employee with two temporal attributes:
   $Employee_histN(\underline{Eid}, (Salary, \underline{Salary_Start}, Salary_End)^*,$
   $(\underline{Hobby, Hobby_Start}, Hobby_End)^*)$
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:
   $SP_histN(\underline{Sid, Pid}, (Price, \underline{Price_Start}, Price_End)^*)$
4. Temporal relationship type SPJ:
   $SPJ_histN(\underline{Sid, Pid, Jid}, (Quantity, \underline{SPJ_Start}, SPJ_End)^*)$
5. Temporal relationship type EmpDep with temporal attribute Position:
   $EmpDep_histN(\underline{Eid, Did}, ((Position, \underline{Position_Start}, Position_End)^*,$
   $\underline{EmpDep_Start}, EmpDep_End)^*)$

□

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

4 Maintaining Data Consistency

Since we separate the current and historical data, we need to ensure consistency when updates occur. Figure 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. Figure 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.

Employee_curr

Eid	Ename	Phone	Salary	Salary_Start
e01	Alice	90000001	8,000	2012-01-01
e03	Smith	90000003	7,000	2014-01-01
e04	Brown	90000004	6,000	2015-01-01

Project_curr

Jid	Tittle	Budget	Project_Start
j04	Deep Learning	80,000	2015-01-01
j05	Temporal Database	60,000	2016-01-01

EmpProj_curr

Eid	Pid
e01	j04
e03	j04
e03	j05
e04	j05

EmployeeHobby_curr

Eid	Hobby	Hobby_Start
e01	Running	1998-01-01
e01	Swimming	2000-01-01
e03	Running	2005-01-01
e04	Reading	2007-01-01

EmpDep_curr

Eid	Did	Position	Position_Start	EmpDep_Start
e01	d01	Manager	2012-01-01	2000-01-01
e03	d01	Engineer	2005-01-01	2012-01-01
e04	d02	Technical Support	2010-01-01	2013-01-01

Department_curr

Did	Dname
d01	Research and Development
d02	Product

EmployeeHobby_hist

Eid	Hobby	Hobby_Start	Hobby_End
e01	Photography	1998-01-01	2005-01-01
e03	Basketball	2006-01-01	2011-01-01
e04	Tennis	2008-01-01	2010-01-01

EmployeeSalary_hist

Eid	Salary	Salary_Start	Salary_End
e01	5,000	2000-01-01	2010-01-01
e01	7,000	2010-01-01	2012-01-01
e03	6,000	2012-01-01	2014-01-01
e04	5,000	2013-01-10	2015-01-01

EmpDep_hist

Eid	Did	EmpDep_Start	EmpDep_End
e01	d02	2000-01-01	2005-01-01
e03	d02	2010-01-01	2012-01-01
e04	d01	2010-01-01	2013-01-01

EmpDepPosition_hist

Eid	Did	Position	Position_Start	Postion_End
e01	d02	Engineer	2000-01-01	2005-01-01
e01	d01	Engineer	2005-01-01	2010-01-01
e01	d01	Senior Engineer	2010-01-01	2012-01-01
e04	d02	Engineer	2007-01-01	2010-01-01

Project_hist

Jid	Tittle	Budget	Project_Start	Project_End
j01	Distributed System	80,000	2007-03-01	2009-01-01
j02	Data Mining	50,000	2008-05-20	2010-08-01
j03	Keyword Search	70,000	2012-10-01	2014-01-01

Fig. 3. Generated schema where current and historical data are separated.

Employee_curr

Eid	Ename	Phone	Salary	Salary_Start
e01	Alice	*90000011*	*10,000*	*2017-06-01*
e03	Smith	90000003	7,000	2014-01-01
e04	Brown	90000004	6,000	2015-01-01

EmployeeSalary_hist

Eid	Salary	Salary_Start	Salary_End
e01	5,000	2000-01-01	2010-01-01
e01	7,000	2010-01-01	2012-01-01
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.

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 <*e01, Swimming, 2000-01-01*> from *EmployeeHobby_curr* in Fig. 3 with a valid end time 2017-08-01, we will insert a tuple <*e01, Swimming, 2000-01-01, 2017-08-01*> 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, *Alice* leaves the company and we delete tuple <*e01, Alice, 90000001, 8000,2012-01-01*> from *Employee_curr* in Fig. 3. This triggers the deletion of all tuples with $eid = e01$ from both current and historical relations. In contrast, suppose 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 tuples involving $j04$ from current relations and insert the corresponding tuples with valid end times in historical relations. If employee is a temporal entity type, then we need to create historical relations to store the non-temporal single-valued and multivalued attributes of employees who have left the company, i.e., they have been deleted from 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 <*e03, d01, 2012-01-01, 2017-09-01*> into the historical relation *EmpDep_hist* and a tuple <*e03, d01, Engineer, 2005-01-01, 2017-09-01*> is inserted into the historical relation *EmpDepPosition_hist*. The tuple <*e03, d01, Engineer, 2005-01-01, 2012-01-01*> 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. Three schemas are generated based on the Employee database[1] which captures 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.

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

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

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

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

$Dept_currN(\underline{deptno}, name)$
$Employee_currN(\underline{empno}, birthdate, name, gender, title, title_Start, salary, salary_Start, Employee_Start)$
$Workfor_currN(\underline{empno}, deptno, Workfor_Start)$
$Manage_currN(\underline{empno}, deptno, Manage_Start)$
$Employee_histN(\underline{empno}, (birthdate, name, gender, (title, \underline{title_Start}, title_End)^*, (salary, \underline{salary_Start}, salary_End)^*, \underline{Employee_Start}, Employee_End)^*)$
$Workfor_histN(\underline{empno}, deptno, (\underline{Workfor_Start}, Workfor_End)^*)$
$Manage_histN(\underline{empno}, deptno, (\underline{Manage_Start}, Manage_End)^*)$

We have 3 datasets for each schema: original Employee dataset D_1, and two synthetically generated datasets D_2 and D_3. Table 1 shows the statistics of these datasets. In D_2, each current entity/relationship has 10 historical entities/relationships, and each temporal relationship/attribute is associated with one time period. This increases the size of historical data 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 increases the

[1] https://dev.mysql.com/doc/employee/en/.

number of repeating groups to evaluate our approach that stores historical data in nested relations. 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.

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	∼1 : 3.2	∼3.6 : 1
D_2	583 MB	10 : 1	1 : 1
D_3	699 MB	1 : 1	10 : 1

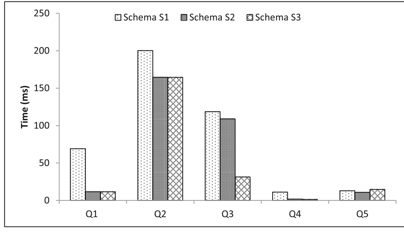
Table 2. Queries for Employee database

Q_1	For 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 $Eid < 100000$ who are currently working in the "Marketing" department
Q_4	Find the job titles of female employees with $Eid < 1000$ 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 that resigned employees (born before 1960-01-01) had joined
Q_8	Find employees who were previously "Engineer" but are now "Senior Engineer"
Q_9	Find employees who had previously managed the department "Marketing"

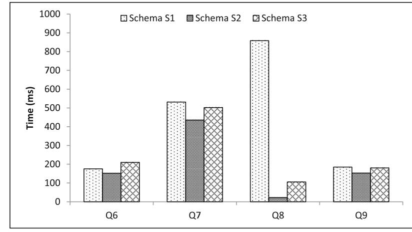
Q_1 and Q_2 query the current data only. These queries run much faster on all 3 datasets with schema $S2$ and $S3$ compared to $S1$. The gap in their runtime widens when the ratio of historical to current data increases (as in D_2). The query times for $S2$ and $S3$ are the same since they have the same 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 $Q5$ 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.

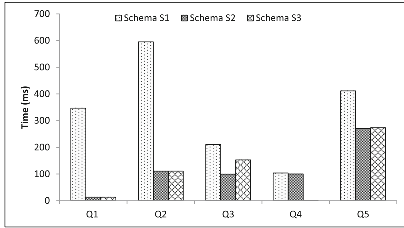
Q_6 to Q_7 are queries on temporal relationships or constrain temporal attributes with some values. As the queries focus on either historical or current data, $S1$ is slower that $S2$. $S3$ is expensive since it is difficult to retrieve entities or relationships whose attributes satisfy some constraints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



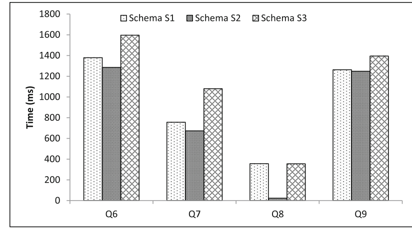
(a) Q_1 to Q_5 on original dataset D_1



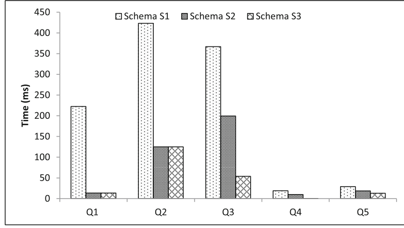
(b) Q_6 to Q_9 on original dataset D_1



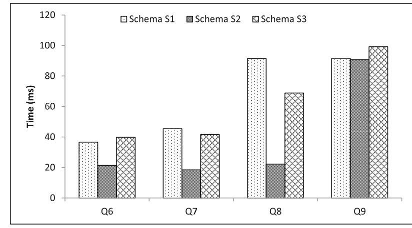
(c) Q_1 to Q_5 on synthetic dataset D_2



(d) Q_6 to Q_9 on synthetic dataset D_2



(e) Q_1 to Q_5 on synthetic dataset D_3



(f) Q_6 to Q_9 on synthetic dataset D_3

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

Q_8 involves 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". This is time consuming. Separating current and historical data avoids 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 runtimes for this query is similar for all three schemes since we need to scan 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 [1,3,9,16–18], or extending existing constructs to include temporal semantics [2,11]. Our ERD^T is a simplified temporal ER diagram that contains the essential constructs for our mapping algorithm.

There are two main approaches to generate a temporal database schema from an ER diagram. One approach is to map a traditional ER model to a database schema, annotate it with temporal functional dependencies, and use the decomposition approach to obtain a set of normal form relations including temporal relations [14]. However, these relations do not capture the semantics of temporal entity, relationship and attribute. Further, when the database schema becomes complex, it is not easy for users to specify the temporal functional dependencies compared to annotating an ER diagram as in our proposed approach.

Another approach is to map a temporal ER model to a temporal database schema. The works in [16,18] map their temporal ER models to the standard ER model first (which may include adding explicit time period attributes) before translating the ER diagram to the database schema. However, 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. The works in [7,11,15,17] map their temporal ER models directly to a database schema. [17] generates a set of 3NF database schema that does not capture the semantics of temporal attributes since they use a temporal ER model which does not support temporal attribute and attributes of relationship type. [7] maps a temporal ER diagram to a surrogate-based relational model (RM/T model) which does not capture the semantics of temporal relationship type. [11] adds an implicit valid time period for each generated relation, thus making all the attributes in the database temporal. This may increase the data redundancy when there are one or more temporal attributes in a relation. All these methods do not separate the current data from historical data.

Although commercial databases such as Microsoft SQL server and IBM DB2 10 implemented the feature of historical and current tables, a historical table is a mirror of its current table, and the purpose is to track modifications in the current table. Further, these tables are only available to data involving transaction time, and not data involving valid time. Moreover, SQL Server and DB2 cannot distinguish between temporal attributes and temporal entities, which needs to be handled differently when updates occur. In contrast, 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 proposed a semantic framework 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 a temporal ER diagram. This accelerates querying of current data, and provides the flexibility of mapping to a set of nested relations. We discussed how consistency between current and historical data is maintained during updates. Experiments showed that the schemas obtained provide efficient access to both current and historical information. Future work includes studying the physical design over our temporal database schema, e.g., temporal indexing.

References

1. Elmasri, R., El-Assal, I., Kouramajian, V.: Semantics of temporal data in an extended ER model. In: ER (1990)
2. Elmasri, R., Wuu, G.T.J.: A temporal model and query language for ER databases. In: IEEE ICDE (1990)
3. Ferg, S.: Modelling the time dimension in an entity-relationship diagram. In: ER (1985)
4. Gao, Q., Lee, M.L., Ling, T.W., Dobbie, G., Zeng, Z.: A semantic framework for designing temporal SQL databases. Technical report, TRB3/18, NUS (2018)
5. Gregersen, H., Jensen, C.: Conceptual modeling of time-varying information. Technical report, TimeCenter TR-35 (1998)
6. Gregersen, H., Jensen, C.S.: Temporal entity-relationship models - a survey. IEEE TKDE **11**, 464–497 (1999)
7. Gregersen, H., Mark, L., Jensen, C.S.: Mapping temporal ER diagrams to relational schemas. Technical report, TimeCenter TR-39 (1998)
8. Khatri, V., Ram, S., Snodgrass, R.T., et al.: Capturing telic/atelic temporal data semantics: Generalizing conventional conceptual models. IEEE TKDE **26**, 528–548 (2014)
9. Klopprogge, M.R.: TERM: an approach to include time dimension in the entity-relationship model. In: ER (1981)
10. Kulkarni, K., Michels, J.E.: Temporal features in SQL: 2011. Sigmod Rec. **41**, 34–43 (2012)
11. Lai, V.S., Kuilboer, J.P., Guynes, J.L.: Temporal databases: model design and commercialization prospects. ACM SIGMIS Database **25**, 6–18 (1994)
12. Ling, T.W.: A normal form for entity-relationship diagrams. In: ER (1985)
13. Ling, T.W., Yan, L.: NF-NR: a practical normal form for nested relations. J. Syst. Integr. **4**, 309–340 (1994)
14. Papazoglou, M., Spaccapietra, S., Tari, Z.: Advances in Object-Oriented Data Modeling. MIT Press, Cambridge (2000). Chap. 7
15. Snodgrass, R.T.: Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, San Francisco (2000). Chap. 11
16. Tauzovich, B.: Towards temporal extensions to the entity-relationship model. In: ER (1991)
17. Theodoulidis, C., Loucopoulos, P., Wangler, B.: A conceptual modelling formalism for temporal database applications. Inf. Syst. **16**, 401–416 (1991)
18. Zimanyi, E., Parent, C., Spaccapietra, S., Pirotte, A.: TERC+: a temporal conceptual model. In: International Symposium Digital Media Information Base (1997)