THE NATIONAL UNIVERSITY
of SINGAPORE

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

TRA3/18

Analyzing Temporal Keyword Queries for Interactive Search over Temporal Databases

Qiao Gao, Mong Li Lee, Tok Wang Ling, Gillian Dobbie and Zhong Zeng

March 2018

T e c h n i c a l R e p o r t

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

Analyzing Temporal Keyword Queries for Interactive Search over Temporal Databases

Qiao Gao[1], Mong Li Lee[1], Tok Wang Ling[1], Gillian Dobbie[2], Zhong Zeng[3]
[1]National University of Singapore, [2]University of Auckland
[3]Data Center Technology Lab, Huawei
{*gaoqiao, leeml, lingtw*}*@comp.nus.edu.sg*;
g.dobbie@auckland.ac.nz; *zengzhong4@huawei.com*

Abstract. Querying temporal relational databases is a challenge for non-expert database users, since it requires users to understand the semantics of the database and apply temporal joins as well as temporal conditions correctly in SQL statements. Traditional keyword search approaches are not directly applicable to temporal relational databases since they treat time-related keywords as tuple values and do not consider the temporal joins between relations, which leads to missing answers, incorrect answers and missing query interpretations. In this work, we extend keyword queries to allow the temporal predicates, and design a schema graph approach based on the Object-Relationship-Attribute (ORA) semantics. This approach enables us to identify temporal attributes of objects/relationships and infer the target temporal data of temporal predicates, thus improving the completeness and correctness of temporal keyword search and capturing the various possible interpretations of temporal keyword queries.

1 Introduction

Temporal relational databases enable users to keep track of the changes of data and associate a time period to the temporal data to indicate its valid time period in the real world. Then users can retrieve information by specifying the time period (e.g. find patients who have fever in 2015), or the temporal relationship between the time periods of temporal data (e.g. find patients who have cough and fever *on the same day*). While such queries can be written precisely in SQL statements, it is a challenge for non-expert database users to write the statements correctly since it requires users to understand the temporal database schema well, associate the temporal conditions to the appropriate temporal data, and apply temporal joins between multiple relations.

Keyword queries over relational databases free users from writing complicated SQL statements and has become a popular search paradigm. However, introducing temporal periods in keyword queries may lead to the problems of (a) missing answers, (b) missing interpretations and (c) incorrect answers if the temporal periods are not handled properly, as we will elaborate.

Missing Answers. This issue arises because traditional keyword search engines treat time-related keywords as tuple values. Consider the sample hospital

database in Fig. 1, which records the temperature and symptoms of patients, salary of doctors, and the dates that patients consult doctors. Suppose we issue the keyword query {Patient cough 2015-05-10} to find patients who have cough on 2015-05-10. Traditional keyword search engine will retrieve patient $p1$ since tuple t_{31}:<*p1,cough,2015-05-10,2015-05-13*> in relation PatientSymptom matches the DATE keyword "2015-05-10". However, patient $p2$ is not returned as an answer even though tuple t_{34}:<*p2, cough, 2015-05-07,2015-05-11*> indicates that $p2$ has a cough on 2015-05-10. This is because $p2$ does not have a tuple in the relation PatientSymptom that matches "2015-05-10".

The work in [9] first adapts relational keyword search to temporal relational database by allowing keywords to be constrained by time periods, and temporal predicates such as BEFORE and OVERLAP between keywords. As such, their method will check if "2015-05-10' is contained within the time period of patients' symptom and retrieve both patients $p1$ and $p2$.

Patient

	Pid	Pname	Gender
t_{11}	p1	Smith	Male
t_{12}	p2	Green	Male
t_{13}	p3	Alice	Female

PatientTemperature

	Pid	Temperature	Temperature_Date
t_{21}	p1	36.7	2015-05-10
t_{22}	p1	39.2	2015-05-11
t_{23}	p1	36.3	2015-06-04
t_{24}	p2	36.7	2015-05-07
t_{25}	p2	38.8	2015-07-13
t_{26}	p3	37.2	2015-10-21

PatientSymptom

	Pid	Symptom	Symptom_Start	Symptom_End
t_{31}	p1	cough	2015-05-10	2015-05-13
t_{32}	p1	fever	2015-05-11	2015-05-13
t_{33}	p1	cough	2015-06-03	2015-06-07
t_{34}	p2	cough	2015-05-07	2015-05-11
t_{35}	p2	fever	2015-07-13	2015-07-15
t_{36}	p3	headache	2015-10-19	2015-10-23

Clinic

	Cid	Cname
t_{41}	c1	Internal Medicine
t_{42}	c2	Cardiology

DoctorSalary

	Did	Salary	Salary_Start	Salary_End
t_{61}	d1	8,000	2000-01-01	2004-12-31
t_{62}	d1	10,000	2005-01-01	2012-12-31
t_{63}	d1	12,000	2013-01-01	2016-12-31
t_{64}	d2	8,000	2005-01-01	Now
t_{65}	d2	10,000	2010-01-01	Now

Consult

	Pid	Did	Consult_Date
t_{71}	p1	d1	2015-05-12
t_{72}	p1	d2	2015-05-13
t_{73}	p1	d1	2015-05-15
t_{74}	p2	d1	2015-05-12
t_{75}	p2	d2	2015-07-13
t_{76}	p3	d1	2015-10-21

Doctor

	Did	Dname	Doctor_Start	Doctor_End	Cid
t_{51}	d1	Smith	2000-01-01	2016-12-31	c1
t_{52}	d2	George	2005-01-01	now	c2
t_{53}	d3	John	2010-01-01	now	c2

Fig. 1. Example Hospital Database.

Missing Interpretations. This issue arises because the work in [9] assume that a *time condition* (temporal predicates and time periods) is always associated with the nearest keyword in the query. This may miss other possible interpretations and their answers to the query. Consider the keyword query {Patient Doctor DURING [2015-01-01,2015-01-31]}. Depending on the user search intention, there are two possible interpretations of this query:

- find patients who consulted doctor during January 2015,
- find patients has consulted some doctor who worked in the hospital during January 2015.

By assuming that the time condition "DURING [2015-01-01,2015-01-31]" is associated with the nearest keyword "Doctor" that matches the relation name Doctor with a valid time period [$Doctor\_Start, Doctor\_End$] indicating the work period of doctor in the hospital, the work in [9] will only return answers for the second interpretation, and miss answers for the first interpretation which is more likely the user search intention.

Incorrect Answers. This issue arises when the time periods in a join operation are not handled correctly, in other words, there is no support for temporal join. Suppose we issue the query {Patient temperature fever DURING [2015-05-01,2015-05-31]} to find the temperature of patients who had a fever during May 2015. This requires a temporal join (joining two records if their keys are equal and their time periods intersect [5]) of the relations PatientSymptom and PatientTemperature. The expected result is 39.2, obtained by joining tuples t_{22} and t_{32}, which gives the temperature of patient p_1 who had a fever during May 2015. The work in [9] only applies the time condition to the nearest keyword "fever" without considering the intersection of time periods during the join operation. Then tuples t_{21} and t_{23} are also joined with tuple t_{32}, adding temperatures 36.7 and 36.3 to the results, which are incorrect because they are not associated with the fever that p_1 had in May 2015.

In this work, we generalize the syntax for temporal keyword queries which comprises of basic keywords and temporal keywords. We design a semantic approach to process complex temporal keyword queries involving temporal joins, taking into consideration the various ways a time condition can be applied. We use an Object-Relationship-Mixed (ORM) schema graph to capture the semantics of objects, relationships and attributes in the temporal databases. With this, we can generate a set of initial query patterns to capture the interpretations of the basic keywords of a query. Then we infer the target time period of the temporal predicate and generate temporal constraints to capture the different interpretations of temporal keywords including an interpretation involving temporal join. We propose a two-level ranking scheme for the different interpretations of a temporal query, and develop a prototype system to support interactive keyword search over a temporal database. Finally, a set of SQL statements is generated from the user-selected query patterns with the temporal constraints translated into temporal joins or select conditions correctly. Experiments on two datasets show the effectiveness of our proposed approach to handle complex temporal keyword queries and retrieve relevant results.

2 Related Work

Existing works on keyword search in relational databases can be classified into data graph and schema graph approaches. In the data graph approach [8, 10, 4, 15, 6], a database is modeled as a graph where each node represents a tuple and each edge represents a foreign key-key reference. An answer to a keyword query is a minimal connected subgraph (Steiner tree) that contains all the keywords. In the schema graph approach [1, 7, 12, 13, 11, 3, 14], a database is modeled as a graph where each node represents a relation and each edge represents a foreign key-key constraint. Based on the schema graph, a keyword query is translated into a set of SQL statements and leverages the RDBMSs to evaluate the statements and retrieve answers. All these works do not distinguish the Object-Relationship-Attribute (ORA) semantics in the database, which leads to incomplete and meaningless results. Moreover, they do not handle time-related

keywords properly and do not support temporal joins between relations, which leads to missing answers and missing interpretations as we have highlighted.

The works in [16, 17] distinguish the ORA semantics and extend keyword queries with meta-data to reduce the ambiguity of keyword queries, and retrieve user intended information and meaningful results. Our work builds upon these works and focuses on identifying the temporal relations in a temporal database and infers the target temporal period of the temporal predicate in the database.

[9] extends keyword queries with temporal constraints and focuses on keyword query efficiency utilizing a data graph approach. However, this work applies the temporal predicate to the nearest keyword in the query and does not consider temporal joins between relations, which leads to missing interpretations and incorrect answers. Besides, without considering the ORA semantics, it also suffers from missing answers and returning incomplete and meaningless results.

3 Preliminaries

Temporal databases support transaction time (time when data is stored) and valid time (time when data is true). In this work, we focus on *valid time* which could be a closed time period or a time point. In addition to augmenting keyword queries with temporal predicates and time periods, we also reduce the ambiguity of queries by allowing users to explicitly indicate their search intention with metadata keywords that match relation/attribute names.

Definition 1. *A temporal keyword query* $Q = \{k_1 \ \cdots \ k_n\}$ *is a sequence of basic and temporal keywords with syntax constraints.*
A ***basic keyword*** *is*

- *a data-content keyword that matches a tuple value, or*
- *a metadata keyword that matches a relation name or an attribute name.*

A ***temporal keyword*** *is*

- *a time period expressed as a closed time period* $[s, e]$ *or time point* $[s]$*, or*
- *a temporal predicate such as AFTER, DURING [2].*

The ***syntax constraints*** *are*

- *the first keyword* k_1 *and the last keyword* k_n *cannot be a temporal predicate,*
- *time periods must be adjacent to a temporal predicate,*
- *for a temporal predicate* k_i*, the previous keyword* k_{i-1} *and the next keyword* k_{i+1} *cannot be a temporal predicate, and* k_{i-1} *and* k_{i+1} *cannot both be time periods.*

The basic keywords specify *what* information users care about, while temporal keywords provide *time conditions* on the information. Temporal predicates are based on [2], and Table. 1 shows their mathematical meanings. The syntax constraints imposed on the keywords ensure meaningful temporal keyword queries. For example, it does not make sense to have a temporal predicate $AFTER$ as the first keyword of a query, and it is meaningless to have a temporal predicate with two time condition operands in a query.

Table 1. Mathematical meaning of temporal predicates

Temporal Predicate	Meaning	Temporal Predicate	Meaning
$[s_1, e_1]$ BEFORE $[s_2, e_2]$	$e_1 < s_2$	$[s_1, e_1]$ AFTER	$s_1 > e_2$
$[s_1, e_1]$ MEETS $[s_2, e_2]$	$e_1 = s_2$	$[s_1, e_1]$ MET_BY $[s_2, e_2]$	$s_1 = e_2$
$[s_1, e_1]$ DURING $[s_2, e_2]$	$s_1 > s_2 \wedge e_1 < e_2$	$[s_1, e_1]$ CONTAINS $[s_2, e_2]$	$s_1 < s_2 \wedge e_1 > e_2$
$[s_1, e_1]$ STARTS $[s_2, e_2]$	$s_1 = s_2 \wedge e_1 < e_2$	$[s_1, e_1]$ STARTED_BY $[s_2, e_2]$	$s_1 = s_2 \wedge e_1 > e_2$
$[s_1, e_1]$ FINISHES $[s_2, e_2]$	$s_1 > s_2 \wedge e_1 = e_2$	$[s_1, e_1]$ FINISHED_BY $[s_2, e_2]$	$s_1 < s_2 \wedge e_1 = e_2$
$[s_1, e_1]$ EQUAL $[s_2, e_2]$	$s_1 = s_2 \wedge e_1 = e_2$	$[s_1, e_1]$ INTERSECT $[s_2, e_2]$	$s_1 \leqslant e_2 \wedge e_1 \geqslant s_2$
$[s_1, e_1]$ OVERLAPS $[s_2, e_2]$	$s_1 < s_2 \wedge$ $s_2 < e_1 < e_2$	$[s_1, e_1]$ OVERLAPPED_BY $[s_2, e_2]$	$e_1 > e_2 \wedge$ $s_2 < s_1 < e_2$

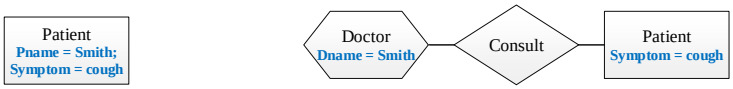
A database can be represented using an Object-Relationship-Mixed (ORM) schema graph $G = (V, E)$. Each node $u \in V$ is an object/relationship/mixed node comprising of an object/relationship/mixed relation and its component relations. An object (or relationship) relation captures the single-valued attributes of objects (or relationships). Multivalued attributes are captured in component relations. A mixed relation contains information of both objects and many-to-one relationships. Two nodes u and v are connected by an undirected edge $(u, v) \in E$ if there exists a foreign key-key constraint from the relations in u to those in v. Fig. 2 shows the ORM schema graph for the database in Fig. 1. Note that an ORM node can have multiple relations, e.g., node Patient contains the object relation *Patient* and two component relations *PatientSymptom* and *PatientTemperature*.

Fig. 2. ORM schema graph of Fig. 1

Based on the ORM schema graph, we can generate a set of query patterns to capture the possible interpretations of the query basic keywords. Details of pattern generation process are in [16]. We illustrate the key ideas with an example.

Example 1 (Query Patterns). Consider the query {Smith cough} which contains basic keywords *Smith* and *cough*. The keyword *Smith* matches some tuple value in relation Patient, while keyword *cough* matches some tuple value in component relation PatientSymptom (see Fig. 1). These relations are mapped to the Patient node in the ORM schema graph in Fig. 2. Based on the matches, we generate the query pattern in Fig. 3(a) which shows an annotated Patient object node.

Another interpretation which finds patients who have a cough and consult doctor Smith is shown in Fig. 3(b). This is because the keyword *Smith* also matches tuple values in the Doctor relation. □



(a) Query pattern P_1 (b) Query pattern P_2

Fig. 3. Query patterns for query {Smith cough}

4 Temporal Query Interpretations

A keyword query that has only basic keywords can be interpreted using the traditional keyword search. However, in temporal databases, we have another interpretation involving temporal join.

Recall that a query pattern P has a set of object/relationship/mixed nodes. We identify the set of temporal relations $\mathcal{S}$ with respect to P that will be involved in a temporal join. A relation R is a *temporal relation* if it has a time period $R[A.Start, A.End]$ or a time point $R[A.Date]$. Here, we also represent a time point $R[A.Date]$ as a time period $R[A.Date, A.Date]$.

For each node $u \in P$, we add the temporal relation $R \in u$ to $\mathcal{S}$ if R is the object/relationship/mixed relation of u, or if R is matched by some query keywords. If $|\mathcal{S}| > 1$, then P has two interpretations. The first interpretation does not consider the temporal aspect of relations in P, i.e., no temporal join or *null* temporal constraint. The second interpretation involves a temporal join between all the temporal relations $R_1, R_2, \cdots, R_m$ in S, indicated by a temporal constraint that restricts the temporal objects, relationships and attributes in P to the same time periods:

$$R_1[A_1.Start, A_1.End] \quad \text{INTERSECT} \quad R_2[A_2.Start, A_2.End] \quad \text{INTERSECT} \cdots \quad R_m[A_m.Start, A_m.End]$$

In other words, we can generate a set of temporal constraints for each query pattern. One query pattern with one temporal constraint forms one complete interpretation of a keyword query.

Example 2 (Temporal constraints). Fig. 4 shows a query pattern P_3 for the query {Patient cough Doctor}. Keyword *Doctor* matches the name of the temporal relation *Doctor* in Doctor node, while keyword *cough* matches some tuple values in the temporal relation *PatientSymptom* in Patient node. The set of temporal relations $\mathcal{S} = \{Doctor, Consult, PatientSymptom\}$.

Table 2 shows the two temporal constraints generated to interpret P_3. One interpretation has a *null* temporal constraint TC_{11} and finds the patients who had a cough and consulted a doctor without any consideration of time. Another interpretation has the temporal constraint TC_{12} and finds the patients who consulted a doctor when they had a cough, which requires temporal joins of the relations in $\mathcal{S}$. □

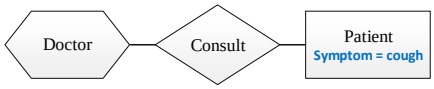


Fig. 4. Query pattern P_3

Table 2. Temporal constraints for {Patient cough Doctor} w.r.t. P_3 in Fig. 4

TC_{11}	*null*
TC_{12}	Doctor[Doctor_Start,Doctor_End] INTERSECT Consult[Consult_Start,Consult_End] INTERSECT PatientSymptom[Symptom_Start,Symptom_End]

On the other hand, when a query has temporal keywords, there is always some temporal predicate TP and the time period may be explicit or implicit.

Queries with Explicit Time Period. Consider the query {Patient cough Doctor DURING [2015-01-01,2015-12-31]} which has a temporal predicate $DURING$ with an explicit time period [2015-01-01,2015-12-31] forming a *time condition*. Fig. 4 shows a query pattern for this query, which is generated without considering the temporal keywords. We can apply the time condition "$DURING$ [2015-01-01,2015-12-31]" to the underlying temporal relations associated with this query pattern in several ways, leading to different interpretations of the query. Table 3 shows all possible interpretations of the time conditions in the form of temporal constraints. Some example interpretations include:

1. (TC_{23}) Apply time condition to temporal relation *Consult* to find patients who had a cough and consulted a doctor during this period.
2. (TC_{24}) Apply time condition to temporal relation *PatientSymptom* to find patients who had a cough during this period and consulted a doctor.

The above interpretations assume the traditional join between the relations that matches the basic query keywords. An additional interpretation is obtained when we apply the time condition after performing a temporal join of the relations. This will find patients who had a cough (during this period) and they consulted a doctor (during this period) who worked in a clinic during this period (TC_{26}).

All the interpretations without temporal join can be obtained by applying the time condition to each temporal relation in a query pattern P. Note that these include temporal component relations in P which are not matched by query keywords, e.g., TC_{22} and TC_{25} in Table 3. The interpretation involving temporal join is obtained by identifying the set of temporal relations $\mathcal{S}$ in P that are involved in the temporal join and applying the time condition to restrict the temporal objects, relationships and attributes in P to the same time periods.

Table 3. Temporal constraints for query {Patient cough Doctor DURING [2015-01-01,2015-12-31]} w.r.t query pattern P_3 in Fig. 4.

TC_{21}	Doctor[Doctor_Start,Doctor_End] DURING [2015-01-01,2015-12-31]
TC_{22}	DoctorSalary[Salary_Start,Salary_End] DURING [2015-01-01,2015-12-31]
TC_{23}	Consult[Consult_Start,Consult_End] DURING [2015-01-01,2015-12-31]
TC_{24}	PatientSymptom[Symptom_Start,Symptom_End] DURING [2015-01-01,2015-12-31]
TC_{25}	PatientTemperature[Temperature_Start,Temperature_End] DURING [2015-01-01,2015-12-31]
TC_{26}	(Doctor[Doctor_Start,Doctor_End] INTERSECT Consult[Consult_Start,Consult_End] INTERSECT PatientSymptom[Symptom_Start,Symptom_End]) DURING [2015-01-01,2015-12-31]

Queries with Implicit Time Period. Consider the query {Patient Doctor AFTER cough } which has a temporal predicate AFTER with no explicit time period. The keyword *cough* matches the temporal relation $PatientSymptom$, and the time period for this query is derived from the tuples that match the keyword *cough*. A query pattern for this query is the same as P_3 in Fig. 4, since these two queries have the same set of basic keywords. Depending on where we apply the time condition, AFTER cough, to the underlying temporal relations associated with this query pattern, we have a number of interpretations, including:

1. (TC_{31}) Apply the time condition to temporal relation *Doctor* to find patients who consulted a doctor who worked in a clinic after the patient had a cough.

2. (TC_{33}) Apply the time condition to temporal relation *Consult* to find patients who consulted a doctor after the patient had a cough.

Note that since a patient could consult doctor several times after s/he had a cough, we may have a set of time periods to consider for the time condition AFTER cough. Here we take the time period with the earliest start time, i.e., the nearest consultation after a patient has cough.

Again, the above interpretations assume the traditional join between the relations that match the basic keywords in the query. We have an additional interpretation when we apply the time condition after performing a temporal join of the relations (TC_{35}). Table 4 shows the temporal constraints obtained. Since the temporal relation *PatientSymptom* (matched by keyword *cough*) is already in the time condition and there is no other keywords matches this relation, we will not apply the time condition to this relation and not include it in the temporal join.

Table 4. Temporal constraints for query {Patient Doctor AFTER cough} w.r.t. query pattern P_3 in Fig. 4.

TC_{31}	Doctor[Doctor_Start,Doctor_End] AFTER PatientSymptom[Symptom_Start,Symptom_End]
TC_{32}	DoctorSalary[Salary_Start,Salary_End] AFTER PatientSymptom[Symptom_Start,Symptom_End]
TC_{33}	Consult[Consult_Start,Consult_End] AFTER PatientSymptom[Symptom_Start,Symptom_End]
TC_{34}	PatientTemperature[Temperature_Start,Temperature_End] AFTER PatientSymptom[Symptom_Start,Symptom_End]
TC_{35}	(Doctor[Doctor_Start,Doctor_End] INTERSECT Consult[Consult_Start,Consult_End]) AFTER PatientSymptom[Symptom_Start,Symptom_End]

Algorithm 1 gives the details of the generation of temporal constraints. A special case occurs when the keywords before and after a temporal predicate matches the same relation, e.g., in the query {Patient Doctor fever AFTER cough}, both keywords fever and cough match the same temporal relation *PatientSymptom*. Fig. 5 shows the corresponding query pattern. In this case, we have one interpretation where we apply the temporal predicate to the temporal relation *PatientSymptom* to find patients who consulted a doctor and had a fever after a cough (TC_{41}), and another interpretation where we apply the temporal predicate after performing a temporal join of the relations (TC_{42}). Table 5 shows the temporal constraints generated.

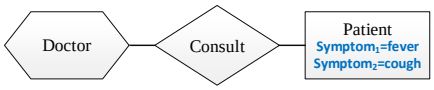


Fig. 5. Query pattern for {Patient Doctor fever AFTER cough}.

Table 5. Temporal constraints for query {Patient Doctor fever AFTER cough} w.r.t. query pattern in Fig. 5.

TC_{41}	PatientSymptom$_1$[Symptom_Start,Symptom_End] AFTER PatientSymptom$_2$[Symptom_Start,Symptom_End]
TC_{42}	(Doctor[Doctor_Start,Doctor_End] INTERSECT Consult[Consult_Start,Consult_End] INTERSECT PatientSymptom$_1$[Symptom_Start,Symptom_End]) AFTER PatientSymptom$_2$[Symptom_Start,Symptom_End]

Algorithm 1: Temporal Constraint Generator

Input: Keyword query Q, a query pattern P for Q
Output: Set of temporal constraints $\mathcal{C}$ for P

$\mathcal{S} = \emptyset$; $\mathcal{T} = \emptyset$; $\mathcal{C} = \emptyset$;
foreach *node* $u \in P$ **do**
  foreach *temporal relation* $R \in u$ **do**
    $\mathcal{T} = \mathcal{T} \cup \{R\}$;
    if *R is the object/relationship/mixed relation in u or R is matched by some keywords* **then**
      $\mathcal{S} = \mathcal{S} \cup \{R\}$;
if *Q has only basic keywords* **then** // Q has no time constraint
  Add $TC_1 = null$ to $\mathcal{C}$;
  if $|\mathcal{S}| > 1$ **then**
    Let $\mathcal{S} = \{R_1, R_2, ...R_m\}$;
    Add $TC_2 =$"$R_1[A_1.Start, A_1.End]$ INTERSECT $R_2[A_2.Start, A_2.End]$ INTERSECT $\cdots$ $R_m[A_m.Start, A_m.End]$" to $\mathcal{C}$;
else if *Q has a temporal predicate k_i* **then**
  if *k_{i+1} is an explicit time period $[s, e]$* **then** // Q has explicit time period
    foreach $R \in \mathcal{T}$ **do**
      Add $TC_R =$ "$R[A.Start, A.End]$ k_i $[s, e]$" to $\mathcal{C}$;
    if $|\mathcal{S}| > 1$ **then**
      Let $\mathcal{S} = \{R_1, R_2, ...R_m\}$;
      Add $TC =$ "($R_1[A_1.Start, A_1.End]$ INTERSECT $R_2[A_2.Start, A_2.End]$ INTERSECT $\cdots$ $R_m[A_m.Start, A_m.End]$) k_i $[s, e]$" to $\mathcal{C}$;
  else // Q has implicit time period
    Let keywords k_{i-1} and k_{i+1} in Q match relations R_{i-1} and R_{i+1} respectively.
    foreach $R \in \{R_{i-1}, R_{i+1}\}$ **do**
      if *R is not a temporal relation* **then**
        Let R' be the temporal relationship relation nearest to R;
        $R = R'$;
    if *R_{i-1} and R_{i+1} have the same relation name* **then**
      Add $TC =$"$R_{i-1}[A_{i-1}.Start, A_{i-1}.End]$ k_i $R_{i+1}[A_{i+1}.Start, A_{i+1}.End]$" to $\mathcal{C}$;
    else
      if *$\mathcal{S}$ and $\mathcal{T}$ contain R_{i+1} that is only matched by keyword k_{i+1}* **then**
        $\mathcal{T} = \mathcal{T} - \{R_{i+1}\}$;
        $\mathcal{S} = \mathcal{S} - \{R_{i+1}\}$;
      foreach $R \in \mathcal{T}$ **do**
        Add $TC_R =$"$R[A.Start, A.End]$ k_i $R_{i+1}[A_{i+1}.Start, A_{i+1}.End]$" to $\mathcal{C}$;
      if $|\mathcal{S}| > 1$ **then**
        Let $\mathcal{S} = \{R_1, R_2, ...R_m\}$;
        Add $TC =$"$(R_1[A_1.Start, A_1.End]$ INTERSECT $\cdots$ $R_m[A_m.Start, A_m.End])$ k_i $R_{i+1}[A_{i+1}.Start, A_{i+1}.End]$" to $\mathcal{C}$;

5 Ranking Temporal Query Interpretations

We have discussed how a temporal keyword query can have multiple query patterns, and each pattern can have multiple temporal constraints depending on how the temporal predicate is applied to the underlying temporal relations. In this section, we describe a two-level ranking mechanism where the first level ranks query patterns without considering the temporal constraints, and the second level ranks the temporal constraints within each query pattern.

For the first level ranking, we adopt the approach in [17]. This work identifies the *target* and *value condition* nodes in a query pattern. A target node specifies the search target of the query, typically the node that matches the first query keyword, while a value condition node is annotated with the attribute value conditions. Query patterns are ranked based on its number of object/mixed nodes and the average distance between the target and value condition nodes. Query patterns with fewer object/mixed nodes and a shorter average distance

are ranked higher. Eqn (1) gives the scoring function for this first level ranking.

$$score_1(P) = \frac{1}{N * \sum_{v \in V} \frac{dist(u, v, P)}{|V|}} \tag{1}$$

where u is the target node, V is the set of value condition nodes and N is the number of object and mixed nodes in query pattern P.

For example, query {Smith cough} has two query patterns P_1 and P_2 (see Fig. 3). The Patient node in P_1 is both a value condition node and a target node. In P_2, Doctor and Patient nodes are value condition nodes, and Doctor node is the target node since the first keyword *Smith* matches doctor's name. P_1 is ranked higher than P_2.

For the second level ranking, we compute a score for each temporal constraint TC of a query pattern P. The temporal constraint with temporal join is ranked the highest since it involves all the temporal relations related to the query. Note that there is at most one temporal constraint with temporal join with respect to one query pattern. For the temporal constraints without temporal join, we first identify the *time condition* node in the query pattern with respect to this constraint. A time condition node contains the temporal relation that the time condition is applied to. There is only one time condition node for each temporal constraint without temporal join. We compute the distance between target node and time condition node in the query pattern. Temporal constraint with smaller distance are ranked higher. The ranking function is shown in Eqn (2).

$$score_2(TC, P) = \begin{cases} 2 & \text{if TC has temporal join} \\ \frac{1}{1 + \ dist(u, t_{TC}, P)} & \text{otherwise} \end{cases} \tag{2}$$

where $u \in P$ is the target node, $t_{TC} \in P$ is the time condition node w.r.t temporal constraint TC. The maximum score for a temporal constraint without temporal join is 1. Temporal constraint with temporal join has a score of 2 so that it is always ranked highest among all constraints.

Note that when the query only contains basic keywords, there are at most two temporal constraints generated (recall Example 2). In this case, we rank the temporal constraint with temporal join first, followed by the null constraint.

Example 3 (Second-level ranking). Consider query {Patient cough Doctor DURING [2015-01-01,2015-12-31]} and its temporal constraints in Table 3 w.r.t. the query pattern P_3 in Fig. 4. TC_{26} has a score of 2 since it involves a temporal join. TC_{21} to TC_{25} have no temporal join, and we compute their scores by finding the minimum distance between target node Patient and the time condition node for each constraint. Both TC_{21} and TC_{22} have a score of $\frac{1}{3}$ since the time condition nodes is *Doctor*, and its distance to the target node Patient is 2. TC_{23} has a score of $\frac{1}{2}$ since the time condition node is node Consult, and its distance to the target node Patient is 1. TC_{24} and TC_{25} have a score of 1 since the time condition node is the same as the target node. □

6 Generating SQL Statements

Finally, we generate a set of SQL statements based on the query patterns and their temporal constraints to retrieve results from the database.

We first consider the query pattern and generate the SELECT, FROM and WHERE clause according to [16]. The SELECT clause includes the attributes of the target node and the FROM clause includes the relations of every node in P. The WHERE clause joins the relations in the FROM clause based on the foreign key-key constraints and translates attribute value condition such as $A = value$ into a selection condition "$contains(R_u.A, value)$". The SQL statement for the query pattern in Fig. 4 for the query {Patient cough Doctor DURING [2015-01-01,2015-12-31]} is as follows. Note that the FROM clause includes relation *PatientSymptom* since it is matched by keyword *cough*.

```
1  SELECT P.*
2  FROM Doctor D, Consult C, Patient P, PatientSymptom PS
3  WHERE D.Did=C.Did AND C.Pid=P.Pid AND P.Pid=PS.Pid
4  AND contains(PS.Symptom,"cough")
```

Next, we consider the temporal constraints of the query pattern. For each temporal constraint of the form of "$R[A.Start, A.End]\ TP\ [s, e]$" where $[s, e]$ is an explicit time period, we translate the temporal predicate TP into a set of comparison operators between $[A.Start, A.End]$ and $[s, e]$ based on Table 1. For example, we translate TC_{24} in Table 3 to the following conditions in the WHERE clause:

"PS.Symptom_Start>2015-01-01 AND PS.Symptom_End<2015-12-31"

For each temporal constraint involving temporal joins, e.g., TC_{26} in Table 3, we first translate the temporal predicate INTERSECT into a set of comparison operator between its adjacent time periods according to Table 1. Then we apply the temporal predicate with time period, like "$TP\ [s, e]$", to one of time periods involved in the temporal join. For example, TC_{26} in Table 3 is translated into conditions as follows, in which lines 5-6 indicate temporal joins and line 7 indicates the temporal predicate with time period.

```
5  D.Doctor_Start ⩽ C.Consult_Date AND D.Doctor_End ⩾ C.Consult_Date AND
6  C.Consult_Date ⩽ PS.Symptom_End AND C.Consult_Date ⩾ PS.Symptom_Start AND
7  PS.PatientSymptom_Start > '2005-01-01' AND PS.PatientSymptom_End < '2005-12-31'
```

7 PowerQ^T System Prototype

Given the inherent ambiguity of keyword queries, we propose to generate various interpretations of the query based on all possible matching of basic keywords and apply the temporal predicate to the different temporal relations. However, it is difficult for users to find the correct interpretation of their query. As such, we design a prototype system called $PowerQ^T$ to allow interactive keyword search over a temporal database. $PowerQ^T$ also includes our two-level ranking mechanism to rank the generated query interpretations, which facilitate users to choose the interpretation that best captures their search intention.

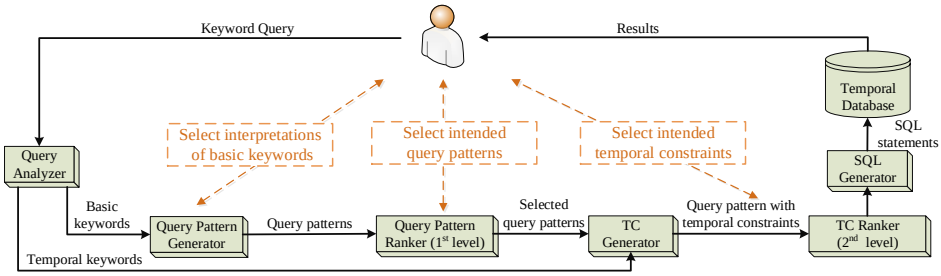


Fig. 6. Architecture of PowerQ^T

Fig. 6 shows the main components of PowerQ^T. Given a keyword query Q, the Query Analyzer distinguishes the basic keywords and temporal keywords in Q. Each basic keyword may have different interpretations as they may have different matches, e.g. keyword *Smith* could be a patient's name or a doctor's name. We allow users to choose the intended interpretations of each basic keyword. Then the Query Pattern Generator generates a set of query patterns based on the selected interpretations of each basic keyword. This reduces the number of query patterns generated. The Query Pattern Ranker uses the first level ranking scheme to rank the generated query patterns for the user to choose. For each selected query pattern, the Temporal Constraint (TC) Generator analyzes the temporal relations and the temporal keywords to generate a set of temporal constraints that depict how the time condition is handled. The Temporal Constraint (TC) Ranker uses the second level ranking scheme to rank the temporal constraints within each query pattern for the user to choose. Finally, we generate SQL statements to retrieve the answers to Q. Note that the answers are grouped by the query interpretations.

This interactive process allows users to consider the interpretations of the basic keywords and temporal keywords separately, and users will not be overwhelmed by too many interpretations.

8 Evaluation

We evaluate the expressive ability of our proposed approach (PowerQ^T) and compare it with the method in [9] (ATQ) which does not consider multiple temporal relations involved in the query and support temporal join. We use the following datasets in our evaluation.

1. *Basketball dataset* [1]. It contains information about NBA players, teams and coaches from 1946 to 2009. We modify the schema to create time period attributes ($from$ and to) based on the original time point attribute ($year$) to make it a temporal database.
2. *Employee dataset* [2]. It contains the job histories of employees, as well as the department where the employees have worked from 1985 to 2003.

[1] `https://github.com/briandk/2009-nba-data/`
[2] `https://dev.mysql.com/doc/employee/en/`

Table 6 shows the schema of these two datasets. A temporal relation is indicated by a superscript T. The DATE type attributes are in *italics*.

Table 6. Dataset schemas

Basketball	**Employee**
Team(tid, location, name)	Department(deptno, dname)
Coach(cid, name)	Employee(empno, ename, gender)
Player^T(pid, name, position, weight, college, *first_season, last_season*)	EmployeeTitle^T(empno, *from*, title, *to*)
	EmployeeSalary^T(empno, *from*, salary, *to*)
PlayerSeason^T(pid, *year*, game, point)	Workfor^T(empno, *from*, deptno, *to*)
TeamSeason^T(tid, *year*, won, lost)	Manage^T(deptno, *from*, empno, *to*)
PlayFor^T(pid, tid, *from*, *to*)	
CoachFor^T(cid, *from*, tid, *to*)	

Table 7 shows the 3 types of queries we designed for each dataset: (a) queries without time constraint, (b) queries with explicit time period, and (c) queries with implicit time period. We evaluate whether PowerQ^T and ATQ are able to retrieve the correct answers with respect to the user search intention.

Type I Queries. These queries do not contain any time constraint, i.e., no explicit temporal predicate or time period (see Table 7(a)). Queries B_1 and E_1 do not involve temporal join, and both PowerQ^T and ATQ retrieve the correct results by matching the query keywords to the database tuples.

Queries $B_2 \sim B_3$ and $E_2 \sim E_3$ involve temporal join and only PowerQ^T could retrieve the correct results. Take for example query B_2. PowerQ^T retrieves the correct results by applying temporal join over the temporal relations *PlayerSeason*, *PlayFor* and *CoachFor* which ensures that only the point history of players who were coached by "Pat Riley" are retrieved. However, ATQ uses the standard join over these temporal relations and also returns the players' point history when they were coached by other coaches.

Type II Queries. These are queries with explicit time period (see Table 7(b)). Queries B_4 and E_4 involves only one temporal relation, and both PowerQ^T and ATQ retrieve the correct results by applying the time period to this relation. However, queries $B_5 \sim B_6$ and $E_5 \sim E_6$ involve multiple temporal relations, and only PowerQ^T retrieves the correct results for them. This is because ATQ does not apply temporal join between relations.

Take for example query B_5. PowerQ^T retrieves the correct results by carrying out a temporal join over the temporal relations *PlayFor* and *CoachFor*, and applying the time condition "OVERLAPS [1990,2000]" to the result of the temporal join. This ensures that we find the coaches for "Magic Johnson" from 1990 to 2000. In contrast, ATQ associates the time period separately to the relations *PlayFor* and *Coachfor*, and returns incorrect results, e.g., "Randy Pfund" is not a correct result since he coached the team "Los Angeles Lakers" from 1992 to 1993, while "Magic Johnson" played for this team only on 1990 and 1995, indicating that Randy did not coach "Magic Johnson" from 1990 to 2000.

Type III Queries. These are queries with implicit time period (see Table 7(c)). Both PowerQ^T and ATQ could retrieve correct results for queries $B_7 \sim B_8$ and

Table 7. Queries for Basketball (B) and Employee (E) datasets

(a) Type I Queries (no time constraint)

#	Query	Keywords	PowerQ^T	ATQ
B_1	Find the position of player "Michael Jordan"	position "Michael Jordan"	✓	✓
B_2	Find the point record of players when they were coached by "Pat Riley"	player point coach "Pat Riley"	✓	✗
B_3	Find how many times the teams won when they were coached by "Phil Jackson"	team won coach "Phil Jackson"	✓	✗
E_1	Find the employee named "Mark" and has been an engineer	employee Mark engineer	✓	✓
E_2	Find the departments where employee "Mark" working as an engineer	department Mark engineer	✓	✗
E_3	Find the department where employees "Mark" and "Leon" are colleagues	department Mark Leon	✓	✗

(b) Type II Queries (with explicit time period)

#	Query	Keywords	PowerQ^T	ATQ
B_4	Find the players who played in team Lakers from 2000 to 2002	player team Lakers OVERLAPS [2000,2002]	✓	✓
B_5	Find the coaches of player "Magic Johnson" from 1990 to 2000	coach "Magic Johnson" OVERLAPS [1990,2000]	✓	✗
B_6	Find the point record of players when they was coached by "Phil Jackson" from 1989 to 2003	point coach "Phil Jackson" OVERLAPS [1989,2003]	✓	✗
E_4	Find the salary from 2008 to 2009 for employee Jackson	salary OVERLAPS [2008,2009] employee Jackson	✓	✓
E_5	Find the salary from 2008 to 2009 for employees who worked in department finance at that time	salary OVERLAPS [2008,2009] department finance	✓	✗
E_6	Find the managers of employee "Danny" in 2005	employee manage Danny OVERLAPS [2005,2005]	✓	✗

(c) Type III Queries (with implicit time period)

#	Query	Keywords	PowerQ^T	ATQ
B_7	Find the number of win times of a team before coach "Phil Jackson" joined in	team won BEFORE coach "Phil Jackson"	✓	✓
B_8	Find the points of players before they coached by "Pat Riley"	player points BEFORE coach "Pat Riley"	✓	✓
B_9	Find the player who played for team Cavaliers then moved to team Suns	player Cavaliers MEETS Suns	✓	✗
E_7	Find the employees who had been an engineer before becoming a technique leader	employee engineer BEFORE "technique leader"	✓	✓
E_8	Find the employees's tittle before they who manage department development	employee tittle BEFORE manage development	✓	✓
E_9	Find the employees who worked in department finance then move to department marketing	employee finance MEETS marketing	✓	✗

$E_7 \sim E_8$ since the target relations of the temporal predicate are easily found by matching the adjacent keywords.

However, for queries B_9 and E_9, only PowerQ^T could retrieve the correct results, and no answers are returned by ATQ. This is because ATQ is unable to interpret the temporal predicate in these queries since the keywords adjacent to the temporal predicate match non-temporal relations. In contrast, PowerQ^T interprets the temporal predicate over the query pattern generated by matching the basic keywords, which finds the temporal relationship relations as the operands of the temporal predicate correctly.

Take for example query B_9. The keywords "Cavaliers" and "Suns" match the relation *Team* which is not a temporal relation. PowerQ^T is able to identify the temporal relation *PlayFor* involved in the generated query pattern as the target relation of temporal predicate MEETS. Thus it is able to retrieve the players who played for team "Cavaliers" then playing for team "Suns".

In summary, we have shown that PowerQ^T is able to retrieve the correct answers for all given queries in each dataset, while ATQ is able to return correct results for some of the queries. There are two reasons why PowerQ^T performs better than ATQ. First, PowerQ^T handles the basic keywords and temporal keywords separately, which enable us to identify temporal relations involved in a keyword query which is not explicitly specified by the users, e.g., queries N_9 and E_9. Second, by analyzing the temporal relations involved in a query pattern, PowerQ^T is able to handle keyword queries that require temporal join between relations, which is not considered in ATQ, e.g., queries N_5 and E_5. Besides these two reasons, there is another advantage of PowerQ^T over ATQ. PowerQ^T helps users to reduce the multiple interpretations of one keyword query into some interpretations which match their search intention based on the interactive search and the two-level ranking mechanism. However, ATQ returns the results of all possible interpretations of one keyword query, which requires additional work on the user's part to filter out the results.

9 Conclusion

In this work, we have studied the problem of evaluating keyword query with temporal keywords (temporal predicate and time period) over temporal relational databases. Existing works do not consider temporal join and the multiple interpretations of temporal keywords, which leads missing answers, missing query interpretations, and incorrect answers. We addressed these problems by considering the Object-Relationship-Attribute semantics of the database to identify the temporal attributes of objects/relationships and infer the target temporal data of temporal predicates. After generating an initial set of query patterns, we can infer the target time period of the temporal predicate and generate temporal constraints to capture the different interpretations of a temporal keyword query. We have also developed a two-level ranking scheme and a prototype system to support interactive keyword search. Evaluation of queries over two datasets demonstrate the expressiveness and effectiveness of the proposed approach.

References

1. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.
2. J. F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 1983.
3. P. de Oliveira, A. da Silva, and E. de Moura. Ranking candidate networks of relations to improve keyword search over relational databases. In *ICDE*, 2015.
4. B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, 2007.

5. H. Gunadhi and A. Segev. Query processing algorithms for temporal intersection joins. In *ICDE*. IEEE, 1991.
6. V. Hristidis, H. Hwang, and Y. Papakonstantinou. Authority-based keyword search in databases. *ACM Trans. Database Syst.*, 2008.
7. V. Hristidis and Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *VLDB*, 2002.
8. A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
9. X. Jia, W. Hsu, and M. Lee. Target-oriented keyword search over temporal databases. In *DEXA*, 2016.
10. V. Kacholia, S. Pandit, and S. Chakrabarti. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.
11. M. Kargar, A. An, N. Cercone, P. Godfrey, J. Szlichta, and X. Yu. Meaningful keyword search in relational databases with large and complex schema. In *ICDE*, 2015.
12. F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.
13. Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: top-k keyword query in relational databases. In *SIGMOD*, 2007.
14. L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: The power of rdbms. In *SIGMOD*, 2009.
15. X. Yu and H. Shi. CI-Rank: Ranking keyword search results based on collective importance. In *ICDE*, 2012.
16. Z. Zeng, Z. Bao, T. N. Le, M. L. Lee, and T. W. Ling. ExpressQ: Identifying keyword context and search target in relational keyword queries. In *CIKM*, 2014.
17. Z. Zeng, Z. Bao, M. L. Lee, and T. W. Ling. A semantic approach to keyword search over relational databases. In *ER*, 2013.