Intelligent Natural Language Query Interface for Temporal Databases

Ramasubramanian P and Kannan A

Abstract. In this article, in order to enable a novice user to interact with the temporal database system and simplify the query processing in temporal database system, a Temporal Natural Language Interface(TNLP) has been designed and implemented. Object evolution in temporal databases is interesting, but none of the SQL-like algebraic languages take evolution into account. This work makes use of a temporal event matching approach for querying temporal data and takes care of evolution. In this work, a Temporal Event Matching Language (TEML) based on the concept of cursor has been designed and implemented for the purpose of pattern matching in query processing. This system has been implemented using Java that can be used in any operating system that supports Java Virtual Machine and has been tested with data from the banking domain.

Keywords: Temporal Databases - Query Processing - Temporal Event Matching Language - Natural Language Interface.

1 Introduction

A database that can store and retrieve temporal data, that is, data which depends on time in some way, is termed as a Temporal Database [1].The classical database is generally two dimensional, and contains only current data. The two dimensions are rows and columns that interact with each other at cells containing particular value. Whereas Temporal databases are three-dimensional with time interval as the third dimension. Temporal Databases can also be referred to as time-oriented databases, time-varying databases, or historical databases [1]. A true temporal database is a bi-temporal database that supports both valid time and transaction time. Transaction time is the actual time recorded in the database at which the data is entered and the time is known as the Time-stamp. Time-stamps can include either only the date or both the date and clock time. Time-stamps cannot be changed. The other major type in temporal database is the valid time. Valid time is the actual or real world time at which point the data is valid.

Conventional databases represent the state of an enterprise at a single moment of time. The conventional database holds the snapshot data. There is a growing interest in applying database methods for version control and design management in e-commerce applications, requiring capabilities to store and process time dependent data. Moreover, many applications such as Medical Diagnosis System, Forest Information Systems, Weather Monitoring Systems and Population Statistics Systems have been forced to manage temporal information in an ad-hoc manner and support the storage and querying of information that varies over time. Temporal database holds time varying information, required by the above mentioned applications.

In the present scenario, writing better database queries for databases pertaining to an organization involves a significant amount of time and expertise. It has become a research issue now to increase the service capability of the database systems to help novice users to formulate a query for database access. High-level query languages such as SQL are available in commercial databases. These are easy for those users with thorough understanding of programming concepts, database schema and relational algebra. To help non-expert users to perform query, a natural language front end is required. For those users who feel SQL difficult to use and for novice users who would like to retrieve data without having to learn querying mechanism such as SQL, a temporal Natural Language querying mechanism has been provided to access data from temporal databases. The Natural Language Interface helps the distribution of the thought process from the human query users to the system. Doing so helps reducing the effort spent by the query users in forming the queries.

The remainder of this paper is organized as follows: After introducing preliminary concepts in Section 1, Section 2 provides an overview of related works done on Query Processing and makes a comparison with our model. Section 3 explains Temporal Event Matching Language(TEML) along with suitable examples. In Section 4, the design of an architecture to support Query processing has been described. Analysis of the system is presented with the aid of Flow Chart developed during this Section 5. Finally, Section 7 provides a concluding remarks and outlines future research directions.

2 Related Works

This section outlines the various related works that naive users use to data access. LUNAR involved a system that answered questions about rock samples brought back from the moon. The program used an Augmented Transition Network (ATN) parser and Woods Procedural Semantics but this system does not focus on database access methods. LIFER/LADDER was designed as natural language interface to a database of information about US Navy ships. This system used a semantic grammar to parse questions and query a distributed database. Wenhua(1992) designed an NLP system for Computer Integrated Manufacturing (CIM) databases which are large, diverse and represent completely different concepts, from accounting to computer-aided-design and schedule planning. Commercial products such as ELF (ELF software Co., 1999) with the components English query and English wizard attempt to generate NLP systems 'on the fly' for any database so that new or user-made databases can be queried with a newly made NLP system. The system TEAM provides a transportable natural language interface to databases. It takes natural language queries as input and translates them into a query language called SODA [2]. Another system IRUS provides a transportable natural language interface for a knowledge based system with procedural components that are independent of any particular domain and database structure. Androutsopoulos et. al [3][4] created a system called MASQUE which is a powerful

and portable natural language front end for Prolog databases. It answers written English questions by generating Prolog queries that are evaluated against the Prolog database. A modified version of MASQUE called as MASQUE/SQL maintains the full linguistic coverage of MASQUE and can be used with any database system supporting SQL [5]. Multilingual natural language interface architecture was given by Werner Winiwarter et al [6] which can be used for accessing on-line product catalogs and lets users formulate their queries in their native languages. English, German and Japanese are the languages handled by this system.

Xiang Sean Zhou and Thanas S. Huang [7] provided the query processing method for image retrieval in multimedia databases. Object evolution in a temporal database can be queried using a event matching language presented by Tsz S.Cheng and Shashi.K.Gadia [8]. The language is a pattern matching language based on the concept of cursor borrowed from SNOBOL4 [8]. In this work, we provide a temporal natural language interface to temporal databases using temporal event matching language by expressing queries based on the temporal attributes associated with data, such as transaction time and valid time. Our model supports the following temporal operators for query processing on both valid time and transaction time. Let [a,b] and [c,d] be any two time intervals.

- [a,b] before [c,d] iff $b<c$
- [a,b] after [c,d] iff $a>d$
- [a,b] during [c,d] iff $(a\geq c)$ and $(b\leq d)$
- [a,b] equals [c,d] iff $(a=c)$ and $(b=d)$
- [a,b] adjacent [c,d] iff $(c-b=1)$ or $(a-d=1)$
- [a,b] overlaps [c,d] iff $(a\leq d)$ and $(c\leq b)$
- [a,b] follows [c,d] iff $(a-d=1)$
- [a,b] precedes [c,d] iff $(c-b=1)$

3 Temporal Event Matching Language

Time is evolutionary in nature and objects evolve in the real world with respect to change of time. A temporal element is a subset of the universe of time T. A temporal element can be expressed as a finite union of intervals. In order to query object evolution, a Temporal Event Matching Language (TEML) can be used as a standalone language or it can be interfaced with query languages for querying temporal data. The purpose of a TEML is to find the occurrence of an event in the evolution. In this language, an event is expressed as a pattern called Event expression.

The language borrows the concept of cursor from SNOBOL 4 [8]. A cursor is simply an instant of time. As the cursor moves from one instant to another, the changes in the evolution are observed. In matching an event, one examines the evolution instant wise through the cursor. The set of instants where the matching takes place is accumulated and the accumulated set of instants are the outcome of the matching process. The time instant at which the cursor points to is called the Cursor Time. The default movement is in the increasing value of time that is one

instant at a time. Setting modes can alter the nature of cursor movement. Like, CursorForward sets cursor to move in an increasing value of time and the mode CursorBackward sets cursor to move in decreasing value of time.

Fig. 1 shows the "Salary Change" to explain the change of events.

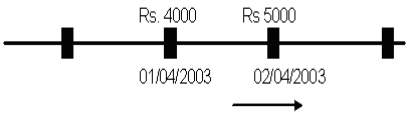


Fig. 1. Salary Change Event

The event "salary went from Rs.4000 to Rs.5000 on 01/04/2003" can be expressed as follows:

moveCursorTo(01/04/2003)
match(salary=4000)
match(salary=5000)

When matching, the system moves the cursor to 01/04/2003. Now the salary is checked to determine if it is Rs.4000. If this the case, then the cursor is moved one instant i.e., to 02/04/2003; otherwise, the match fails. Similarly if the salary is Rs.5000 at the current cursor time then the event is matched and cursor advances to next instant.

In the matching process, the cursor may land beyond the domain of the evolution. In order to avoid this, two constraints like lower fringe and upper fringe are defined for the system. The fringes do not have any predefined values from the starting and the ending of the event domain but it varies according to the system under consideration.

3.1 Temporal Event Matching Language Statements

The Event matching expression is a sequence of statements where the flow of control is similar to a general purpose language. There are specific statements that has been defined for positioning cursor at a specific instant:

- **moveCursorTo(time instant)** allows the cursor to jump to a specific time instant.
- **moveCursorBy(units)** allows the cursor to be advanced by the necessary time units.
- **advanceCursorToMatch(condition)** advances the cursor until the condition is satisfied.

Boolean expressions are also available in this representation. For example, the expression cursorAt(time instant) verifies if the cursor is at the specified time instant.

3.2 Storing Event Expressions In Macros

Sometimes the Event expressions may be complex, in such cases it can be decomposed into logical sub expressions. Such sub expressions are stored in a macro.

Macro syntax:

<macro>::=macro <macro name>(<argument list>)<statements>

Consider the event "Salary raised from Rs.4000 to Rs.5000 on 01/04/2003".

```
macro salaryChange(A:attribute,v1:int,v2:int,t:instant)
{
      moveCursorTo(t);
      match(A=v1);
      match(A=v2);
}
```

Now, the given event is expressed as "salaryChange(salary,4000,5000,01/04/2003)".

When the macro is invoked, the attribute salary, the values 4000, 5000 and the time instant 01/04/2003 are passed to the macro that are bound to the variables A, v1, v2 and t and the event in the body of the macro are matched. There is only one global cursor and only one global event domain being computed and the matching may be either success or failure. Input parameters are replaced by the actual arguments. A macro does not have a context that is independent of the calling Event expression.

3.3 Integration of TEML and SQL

The Temporal Event matching language discussed in this work can act as a standalone language for querying temporal data. It can also be integrated with SQL for querying temporal databases.

For example, to give the empno and salary of the entire employee supporting the TEML macro salaryChange can be expressed as

"Select empno,name from employee during
salaryChange(salary,4000,5000,01/04/2003)"

The macro salaryChange returns the event domain respective to the match and from that the necessary columns are taken.

TEML macros can be used as standalone procedures or can be embedded with SQL statements. The embedded TEML statements are recognized in the query processor and are given to the TEML precompiler for further processing. The TEML precompiler converts the TEML statements into SQL calls and gives them

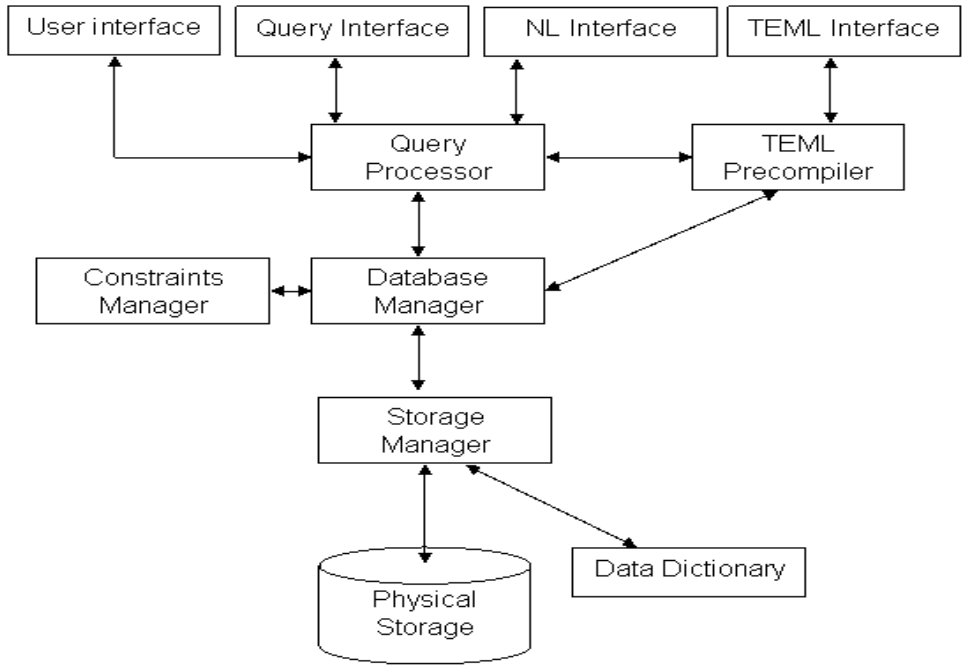


Fig. 2. Temporal Query Processing Architecture

to the query processor. The SQL calls are then given to the table and the resultset is derived and given to the user interface.

4 Architecture

The overall architecture of the Temporal Query Processing and Access Control System is depicted in Fig. 2. It consists of ten basic elements as explained below:

- **User interface:** This part provides a Visual Query Builder for SQL. The input from this is the query and the output is the resultset from the database.
- **Query interface:** This interface provides a mechanism for directly giving the query and getting back the resultset.
- **Natural Language(NL) Interface:** It enables a novice user to interact with the database system. It accepts queries in English and forwards it to the NL Precompiler. The NL precompiler uses the dictionary for transformation of known phrases and then the morphological analysis and syntax analysis are carried out. The syntactically correct query is forwarded to the query analyzer. The query is transformed from natural language question into SQL query.
- **TEML interface:** This is an editor for TEML. The user can directly process the TEML statements from this editor or can store the TEML macros and integrate with SQL. The macros are stored as *.eml files.
- **Query Processor:** This is the component where the query is processed. The macro is passed to the TEML precompiler and the remaining SQL statements

are processed. The process involves steps like lexical analysis, syntactic analysis and semantic analysis. The query processor has to consider the valid time and its granularity before processing the query. The time based authorization constraints are also considered.

- **TEML Precompiler:** The TEML statements are processed here. The input can be either from TEML interface or from query processor. The TEML macro is compiled and event matching takes place here. The resulting event domain is returned to the query processor. The precompiler converts the statements in the macro to events.
- **Database manager:** This defines the datatypes that are to be stored in the database. It also checks for the validity of the data by passing them to the Constraints Manager. It captures the resultset of the query from the physical storage. Optimization techniques in storage and retrieval of information from physical storage are processed here.
- **Constraints Manager:** This holds various rules and constraints required to access the tables. The data is inserted into the table only if the constraints are satisfied.
- **Storage Manager:** This defines various data structures that are to be used to store the data in the physical storage. Log manager is inbuilt in this. The log manager keeps track of all operations that had taken place in the database system and stores them in a log book so that they can be used in rollback and commit operations. Rollback and commit operations can be carried out based on the transaction time. Errors are recorded in an Error Log file.
- **Data Dictionary:** This stores the meta data of the tables like the table name, table type (temporal/non temporal), fields in the table and their types, in the physical storage. There are provisions for locking tables in the data dictionary (0-no lock, 1-shared lock, 2-exclusive lock). A special delimiter separates details of each table. The data dictionary is used for validating the fields before query processing. A separate data dictionary is used for handling temporal and non temporal data.
- **Physical Storage:** The tables are stored permanently in the form of files. These files contain two parts namely the Header part and the Data part; the header contains information's like name of the table, number of fields, their types and constraints and the data part contains the records. The records in the temporal table are time stamped. The time stamp contains the date and time of the transaction. The system is taken as the reference.

4.1 Architecture Of TEML Precompiler

The general architectural framework for the TEML Precompiler is illustrated in Fig. 3. It consists of four main components as represented below.

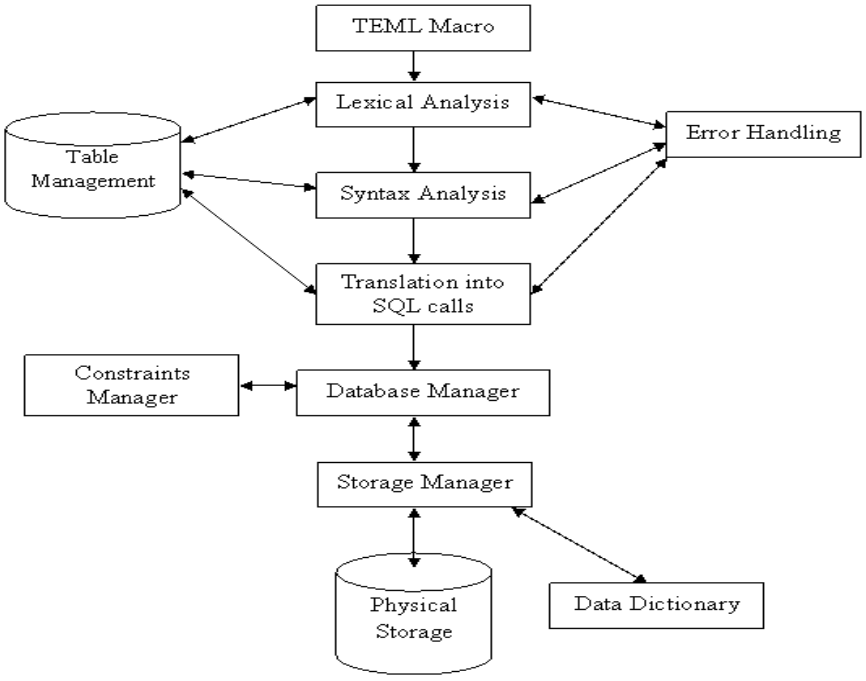


Fig. 3. Architecture of TEML Precompiler

- **TEML Macro Source:** The TEML interface or the query processor forms the TEML source. The macro consists of TEML statements. A TEML call can also come from query processor if the TEML integrated with the SQL statements are invoked.

- **Lexical Analysis:** The TEML macro is classified into stream of tokens and the type of tokens are identified in this phase. The tokens are then stored in tables for the next phase.

- **Syntax Analysis:** This phase checks the syntax of the TEML macro and reports to the error handling system. The syntax analyzer groups tokens into syntactic structures.

- **Translation into SQL calls:** This phase processes the TEML statements and converts them into SQL calls and passes them to the database manager. The database manager in turn takes these statements and gives the resulting database. Constraints manager while processing checks the upper fringe and lower fringe.

5 Flow Chart

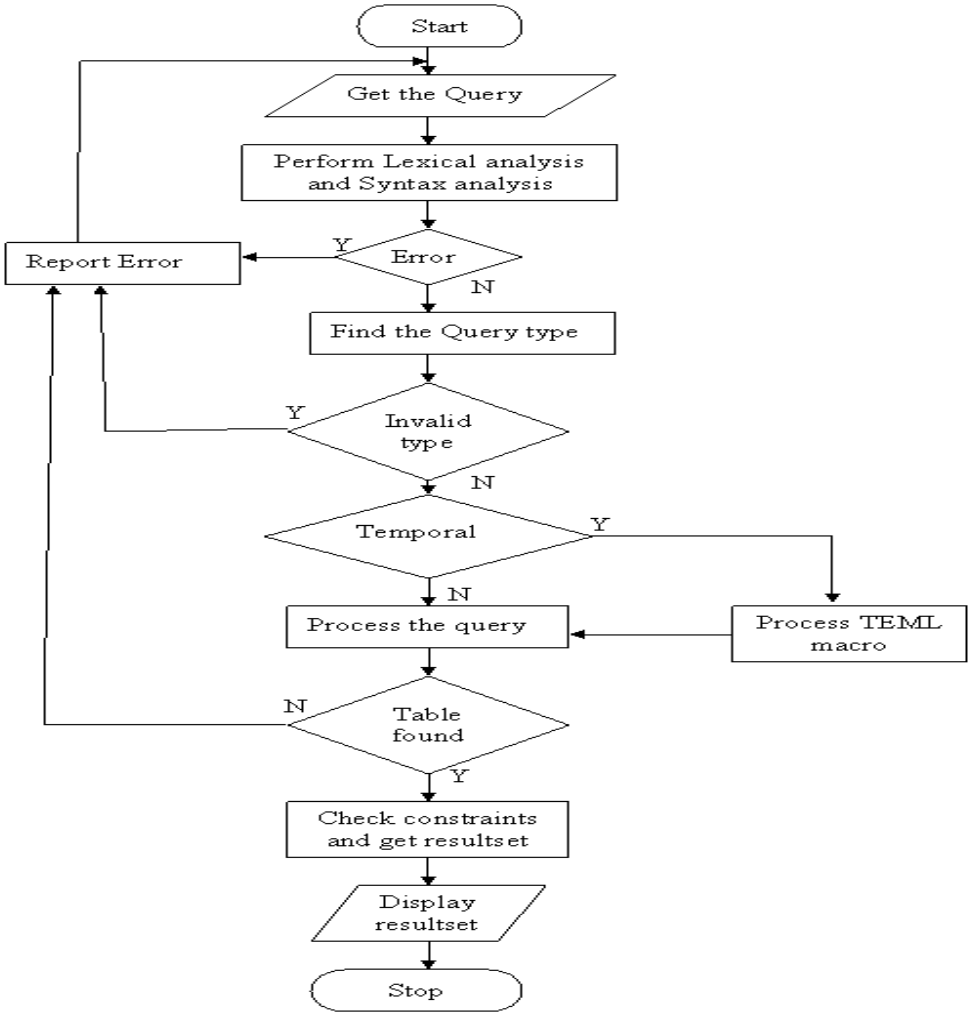


Fig. 4. Flow Chart for Temporal Query processing

The flowchart describes various operations that take place while processing a temporal query. The processing includes phases like lexical analysis, parsing and processing the query. Queries for both temporal and non-temporal tables form the input to the flowchart. The query undergoes the lexical analysis where it is grouped as tokens. The syntax of the query is checked based on the tokens, which is the syntax analysis phase. The error handling is done accordingly. Then the query is processed by the query processor and the type of the query and its arguments are identified. The error is reported for each phase and the process is terminated accordingly. After the query is processed, the results are obtained and they are passed on to the user interface for further operations.

In temporal queries, TEML macros and the arguments are integrated with the query. Hence, before processing the query, the TEML part is separated and passed on to the TEML precompiler where it is being processed. The results from the TEML precompiler are used for further processing of the query. Constraints are also checked during the query processing which the Constraint Manager does. Thus the query processing is a joint activity of Database Manager, Constraints Manager, Storage Manager, Data Dictionary and TEML Precompiler.

6 Results

Temporal table stores both valid and transaction time.

Consider a table emp(EmpNo varchar,Name varchar,Department varchar,Salary float) with validtime(day) and transaction attributes.

EmpNo	Name	Department	Salary	From	To	Time stamp
pd01	Ramkumar	Production	5000	01/03/2003	04/06/2003	28/3/2003 11:34
pd02	Sathish	Production	5000	01/03/2003	05/06/2003	28/3/2003 11:34
pd03	Ravishankar	Production	5000	01/03/2003	05/06/2003	28/3/2003 11:35
mk01	Krishnakumar	Marketing	6000	01/04/2003	31/05/2003	28/3/2003 11:36
mk02	Rajarajan	Marketing	6000	01/04/2003	09/06/2003	28/3/2003 11:36
hr01	Senthilkumar	HRD	8000	01/05/2003	31/12/9999	28/3/2003 11:37
hr02	Vishnu	HRD	7500	01/05/2003	31/12/9999	28/3/2003 11:38
mk01	Krishnakumar	Marketing	8000	01/06/2003	31/12/9999	28/3/2003 11:41
mk03	Karthikeyan	Marketing	6500	04/06/2003	31/12/9999	28/3/2003 11:53
pd01	Ramkumar	Production	7500	05/06/2003	31/12/9999	28/3/2003 11:54
pd02	Sathish	Production	8500	06/06/2003	31/12/9999	28/3/2003 12:01
pd03	Ravishankar	Production	8500	06/06/2003	31/12/9999	28/3/2003 12:04

Table 1. Employee Table

NL Query : Get the name and identification number of the employee who got salary change from 5000 to 8500 on 6/6/2003.

```
Morphological Analysis:
**********************
```

Initial Transformation:
List the name,empno of employee during salaryChange(5000,8500,6/6/2003)

Get v //verb

The d //determiner

Name n //noun

Empno	n //noun
Of	p // pronoun
Employee	n //noun
During	t //temporal operator
salaryChange	m //TEML macro
5000,8500,6/6/2003	u //user defined values

Key Words:
****

Nouns	: name,empno
Nouns - other than field names	: details,employee
Table Name	: emp
Final Field List	: name,empno
Temporal Event	: during
TEML macro	: salaryChange
Condition	: salary = 5000, salary=8500, validtime = 06/06/2003

```
SQL Query:
**********
SELECT * from emp during salaryChange(5000,8500,06/06/2003)

TEML Macro:
***********
salaryChange(int v1,int v2,day d)
{
setTableAs(employee);
setEventDomain(true);
moveCursorToPreviousDay(d);
match(salary,v1);
moveCursorTo(d);
match(salary,v2);
}
```

Results:

pd02	Sathish
pd03	Ravishankar

7 Conclusions and Future Works

In this paper, a Temporal Natural Language Interface to query from temporal databases has been designed and implemented. This system helps the novice users to interact with temporal databases by giving a natural language query. In order to capture events in evolutions, the temporal event matching approach in the form of a pattern recognition language has been implemented in this work. This system recognizes the evolutionary nature of time, more naturally, compared to traditional languages.

Future research in this could be the extension of the Temporal Natural Language Interface, to support various domains using Global Dictionary and have a Multi Linguistic module to work with various languages such as German, Japanese and Tamil. Future advancements in this direction would add up to this system with fuzzy natural language interface to image databases. The TEML can be extended into a general purpose language with more control features, sub programs and built in functions. Time is stored as an object and so the various time related manipulations can be performed in this work. This system returns one single event domain for every macro and the system cannot return multiple domains based on multiple attributes. This work can be extended to provide an automatic translator that can translate an existing snapshot database into a temporal database.

References

1. Tansel, Clifford, Shashi Gadia, and Richard Snodgrass. *Temporal Databases: Theory, Design and Implementation.* Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, second edition, 1993.
2. Grosz J.B. Team: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32:173–243, 1987.
3. Anxerre P and Inder R. *MASQUE Modular Answering System for Queries in English - User Manual.* AI Applications Institute, University of Edinberg, tech. rep. aiai/sr/10 edition, 1986.
4. Androutsopoulos I, Ritchie G, and Thanisch P. Masque/sql - an efficient and portable natural language query interface for relational databases. Database technical paper, Department of AI, University of Edinburgh, 1993.
5. Androutsopoulos I. Interfacing a natural language front end to a relational database. Technical Paper 11, Department of AI, University of Edinburgh, 1993.
6. Winiwarter W and Ismail Khalil Ibrahim. *A Multilingual Natural Language Interface for E-Commerce Applications.* Ph.d thesis, University of Vienna, Austria, 2000.
7. Xiang Sean Zhou and Thanas S. Huang. Unifying keywords and visual contents in image retrieval. *IEEE Transactions on Multimedia*, 2(1):1–13, 2000.
8. Tsz S. Cheng and Shashi K. Gadia. The event matching language for querying temporal data. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1119–1125, 2002.

8 Appendix

8.1 Screenshots

Fig. 5 shows the output in the query interface after a temporal table is created. The query interface is created using Java (awt and swing).

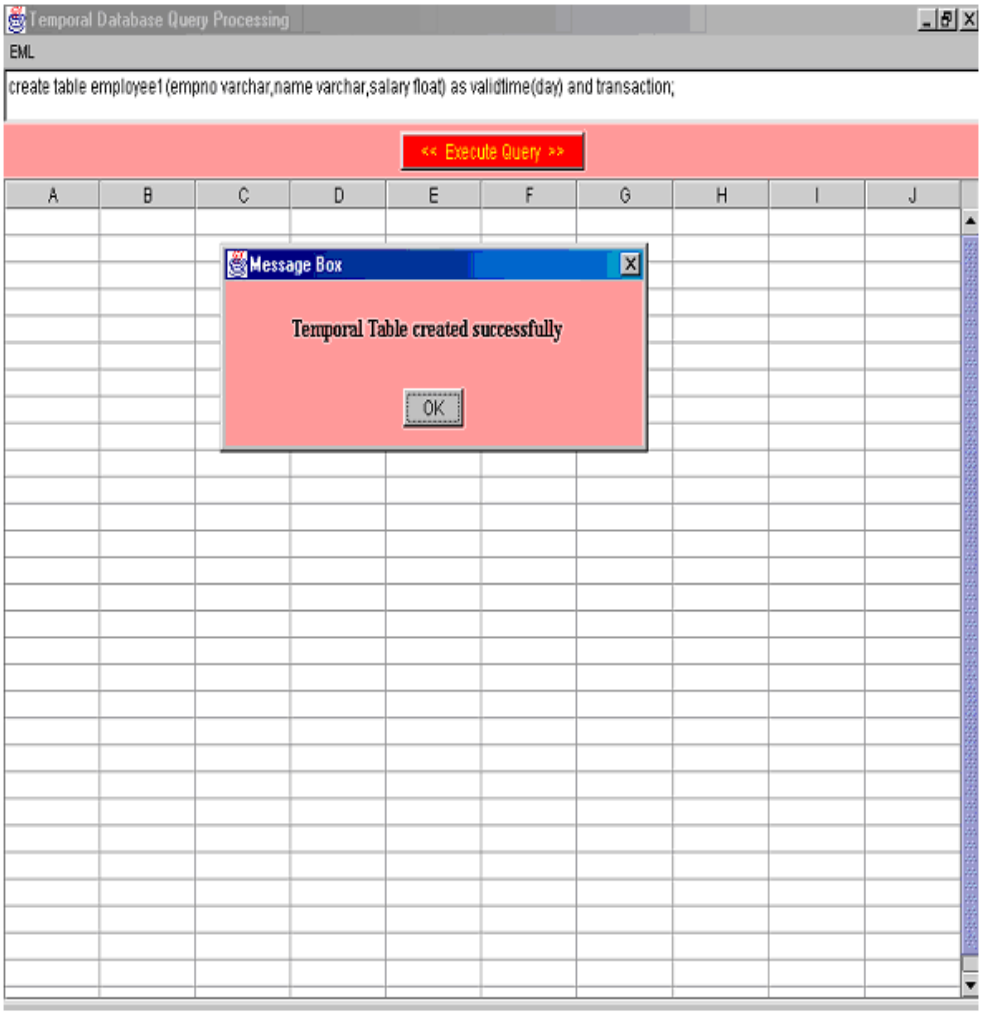
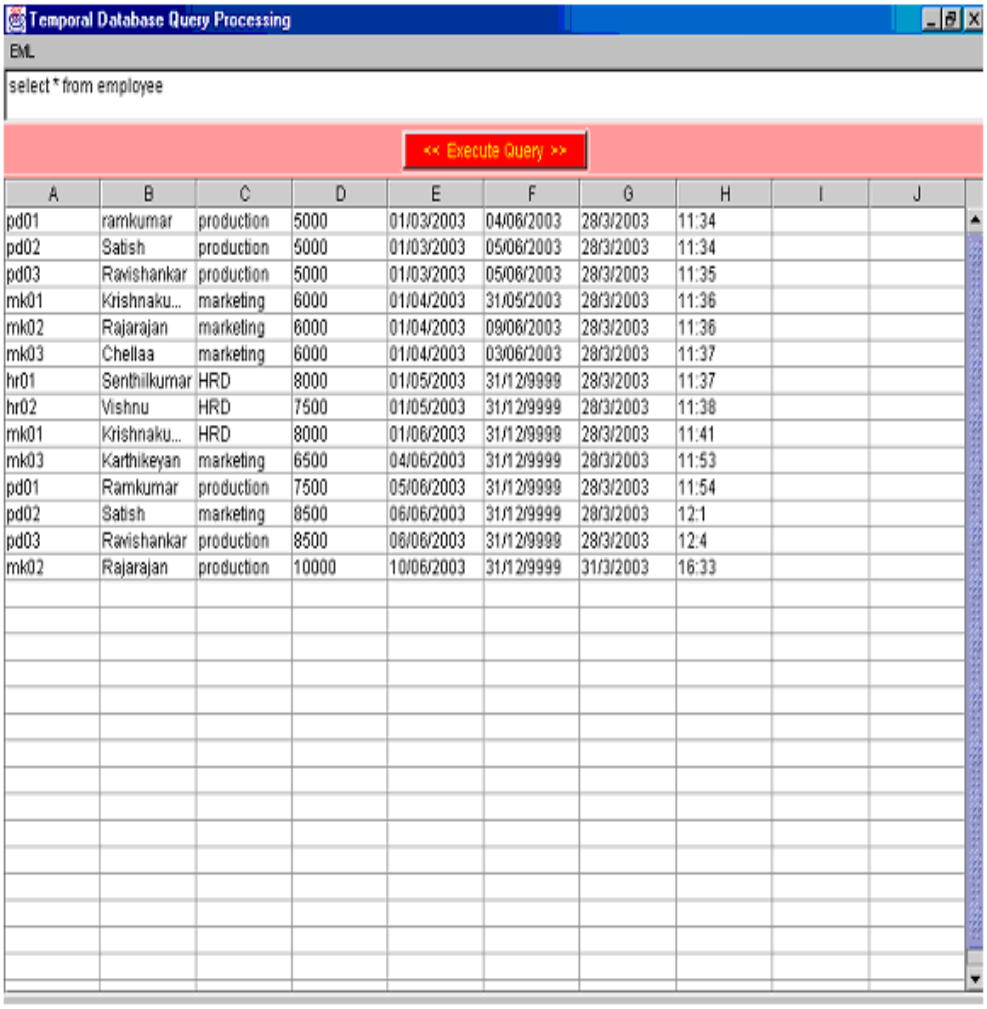


Fig. 5. Query Interface

The output after inserting the rows and a select query is given to the table employee is shown in the following Fig. 6.



A	B	C	D	E	F	G	H	I	J
pd01	ramkumar	production	5000	01/03/2003	04/06/2003	28/3/2003	11:34		
pd02	Satish	production	5000	01/03/2003	05/06/2003	28/3/2003	11:34		
pd03	Ravishankar	production	5000	01/03/2003	05/06/2003	28/3/2003	11:35		
mk01	Krishnaku...	marketing	6000	01/04/2003	31/05/2003	28/3/2003	11:36		
mk02	Rajarajan	marketing	6000	01/04/2003	09/06/2003	28/3/2003	11:36		
mk03	Chellaa	marketing	6000	01/04/2003	03/06/2003	28/3/2003	11:37		
hr01	Senthilkumar	HRD	8000	01/05/2003	31/12/9999	28/3/2003	11:37		
hr02	Vishnu	HRD	7500	01/05/2003	31/12/9999	28/3/2003	11:38		
mk01	Krishnaku...	HRD	8000	01/06/2003	31/12/9999	28/3/2003	11:41		
mk03	Karthikeyan	marketing	6500	04/06/2003	31/12/9999	28/3/2003	11:53		
pd01	Ramkumar	production	7500	05/06/2003	31/12/9999	28/3/2003	11:54		
pd02	Satish	marketing	8500	06/06/2003	31/12/9999	28/3/2003	12:1		
pd03	Ravishankar	production	8500	06/06/2003	31/12/9999	28/3/2003	12:4		
mk02	Rajarajan	production	10000	10/06/2003	31/12/9999	31/3/2003	16:33		

Fig. 6. Visual Query Builder

The following Fig. 7 shows the log manager in action.

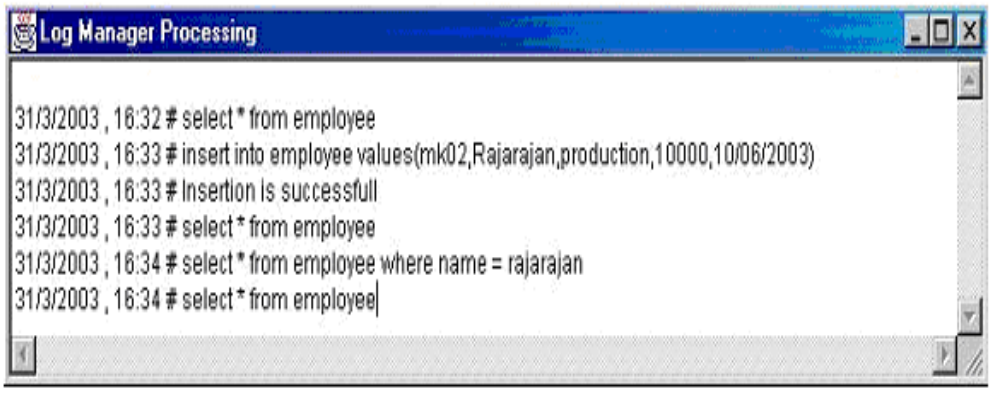


Fig. 7. Log Manager

The following output screen(Fig. 8) gives the look of the visual query builder used for user interface. This user interface is created using swings.

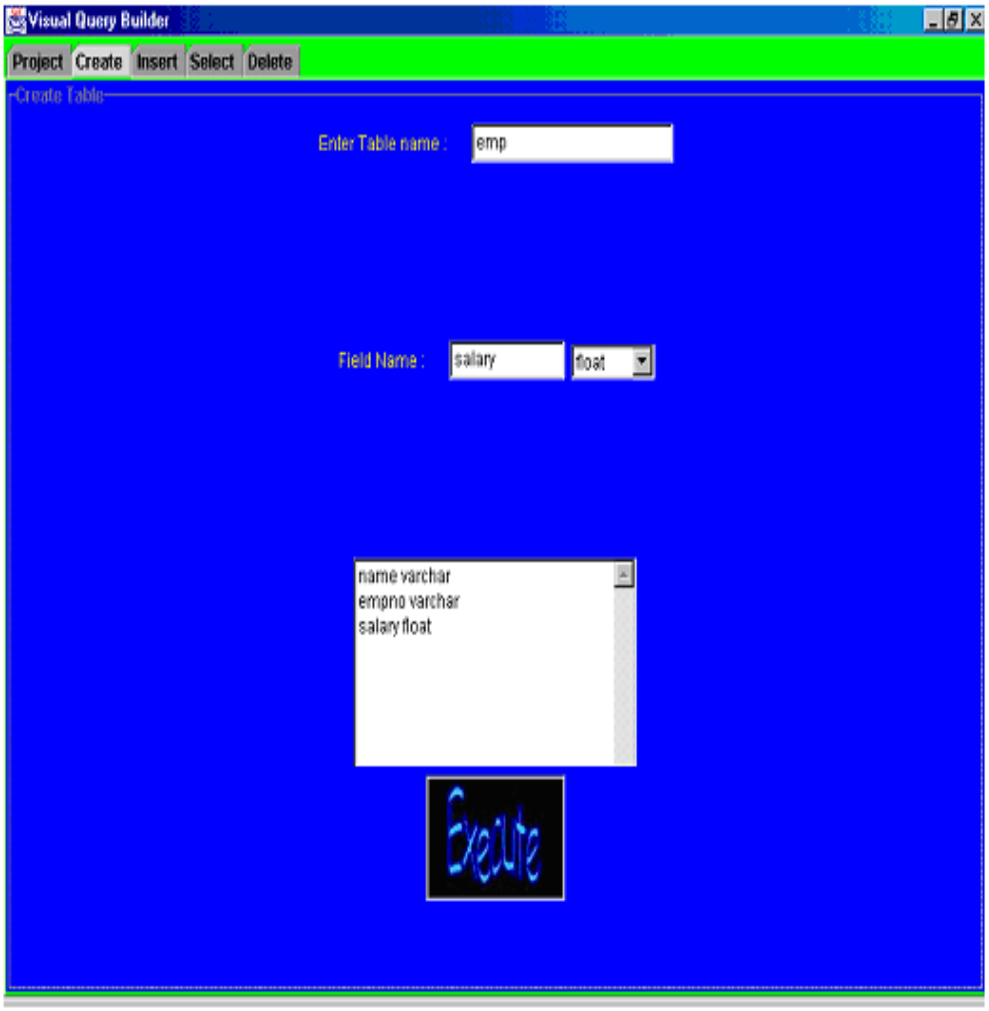


Fig. 8. User Interface

The following screen(Fig. 9) is the view of the TEML Editor. This is also developed in Java (awt)

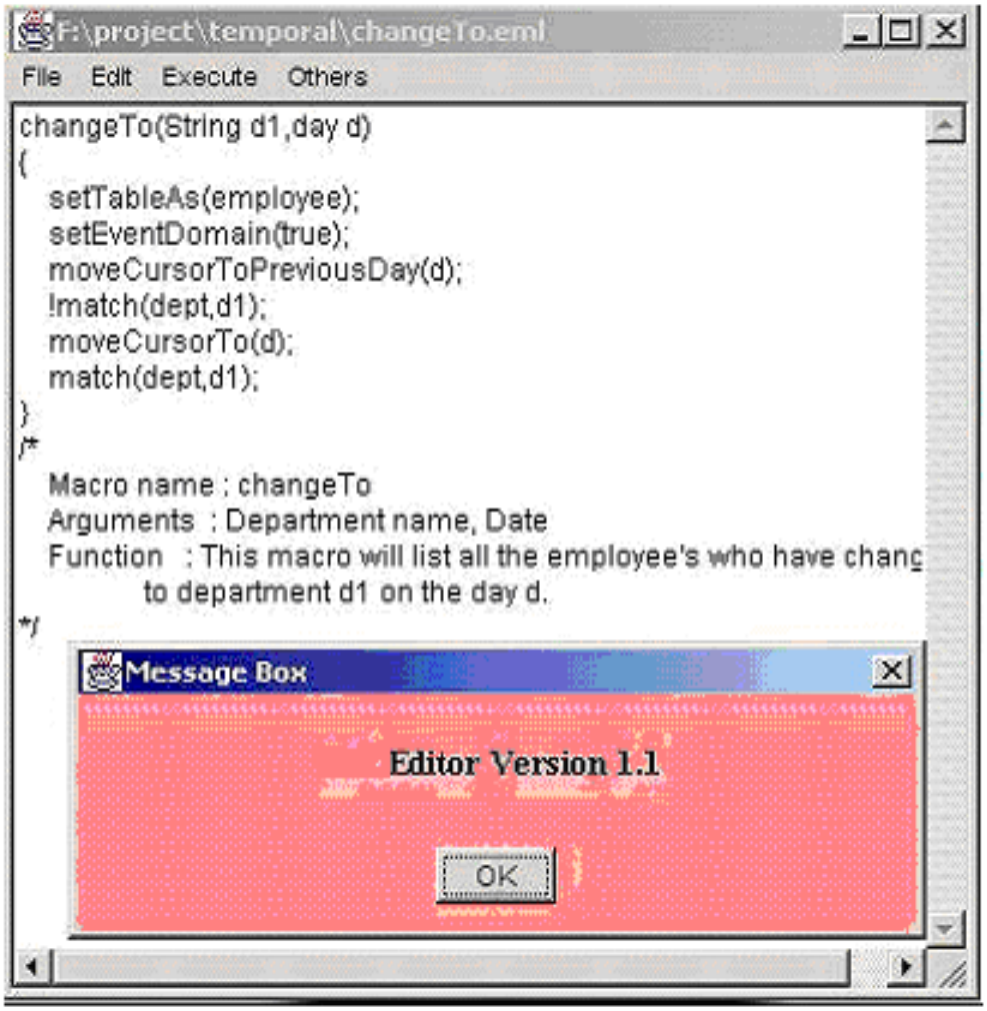


Fig. 9. TEML Editor