CHAPTER 4

TEMPORAL DATA MANAGEMENT AND REASONING

Every database system provides a query language to represent and manipulate data. In the relational databases, the selection of tuples is only based on the attribute values. On the other hand in temporal databases, the selection of tuples is not only based on the attribute values but also on the temporal constraints. Hence, the temporal database system which is designed and implemented in this research work provides a specially designed query language called Extended Temporal Mining Structured Query Language (ETMSQL) for effective data representation and retrieval of temporal data with deductive inference.

4.1 IMPLEMENTATION OF ETMSQL

The query language ETMSQL has been designed for querying the intelligent temporal database management system proposed in this work. This language has been designed by integrating and extending the query languages ETSQL and TQML (Xiaodong Chen et al 1998). ETMSQL provides extensions for temporal data management using instant and interval comparison operators and also provides the TIME SLICE clause. ETMSQL provides additional special constructs for table creation, insertion, updation, deletion and retrieval of tuples with temporal data management features in each statement. Moreover, the SELECT statement used for retrieval provides special features for temporal reasoning with mining options and rule management.
4.1.1 CREATE Statement

The CREATE statement creates a table in the database with a particular table name. The created table contains user defined fields and three additional fields such as STIME, ETIME, and SYSTIME. The STIME represents the starting time of the valid time interval, ETIME represents the ending time of the valid time interval and SYSTIME represents the system time which records the time at which a transaction is carried out by the system. The simplified syntax of the CREATE statement is as follows:

```
CREATE TABLE tablename
    (fieldname1 type (width)
    [,fieldname2 type (width)] ...)
        [ WITH TIME GRANULARITY YEAR];
        type → INTEGER / CHAR / FLOAT /.....
```

For example, the CREATE statement

```
CREATE TABLE EMP
    (ENAME CHAR (5), ENO INTEGER(2))
    WITH TIME GRANULARITY YEAR;
```

creates a table named EMP consisting of five attributes namely ENAME, ENO, STIME, ETIME and SYSTIME.

Whenever a CREATE statement is executed, it creates not only the base table but also it creates its corresponding history table. Then the details regarding the names, attributes and types are stored in the data dictionary.
4.1.2 **INSERT statement**

Values are inserted into a table using an INSERT statement. The syntax of the INSERT statement is:

```
INSERT INTO tablename VALUES
(value1 [, value2] ...);
```

When the user uses the INSERT statement, the existence of the table name is checked from the data dictionary. Moreover, a rule is executed to check the period in which the data values are entered into the table. Finally, the type and width of each field are checked by the system using the entries present in the data dictionary.

For example, the INSERT statement

```
INSERT INTO EMP VALUES
(GANESAN, 1, 20070101, 20070202);
```

inserts a row with the values given in the command and also the system time. Hence, the information for the STIME and ETIME fields should be provided by the user whereas the system time (SYSTIME) is inserted automatically by the system.

4.1.3 **UPDATE statement**

The values present in a table are updated using the UPDATE statement.
The syntax of the UPDATE statement is:

```
UPDATE tablename option
    SET attribute1 = newvalue1 [ , attribute2 = newvalue2 ] ...
    [ WHERE selection_condition ] ;
```

Here, the option refers to either CORRECTION or MODIFICATION. The CORRECTION option is used whenever the values entered by the user are not needed for maintaining the history. They can be changed into a new value using this CORRECTION option. In this case, the previous values are not written into the history table. But in the MODIFICATION option, the previous values are written into the history table and the modified values are stored in the base table. The history table maintains the history of updates that are carried out on the base table.

For example, the UPDATE statement

```
UPDATE EMP CORRECTION
    SET NAME = 'HEMA'
    WHERE ID = 1;
```

updates the EMP table without storing the old values in the history table. On the other hand, if the user uses the MODIFICATION option, the previous values are transferred to the history table and the current data values are stored in the base table.

### 4.1.4 Delete Statement

The values in the table are deleted using the DELETE statement.
The syntax of the DELETE statement is:

```
DELETE FROM tablename option  
[ WHERE selection_cond ] ;
```

Here, the option is either general delete (CORRECTION) or MODIFICATION.

For example, the DELETE statement

```
DELETE FROM EMP MODIFICATION  
WHERE NAME = 'KANNAN';
```

deletes the records having the name ‘KANNAN’ from the base table but stores in them in the history table.

### 4.1.5 SELECT statement

The purpose of the SELECT statement in ETMSQL is to retrieve data from the base table as well as history table and to search for derived formulae learnt from previous queries. It also initiates the temporal reasoning tasks through WHEN, WITH, MWHERE and MWITH clauses.

The syntax of the SELECT statement is:

```
SELECT   <attribute_list>  
FROM    <table_list>  
[ WHERE selection_cond ]  
[ TIME_SLICE [NUM, NUM ]]  
[ FOR field < instant_operator > value ]  
[ WHEN < interval_operator > [NUM, NUM]]
```
[WITH FORECAST/ANALYSE/PLAN [AND LEARN] IDENT ON NUM];
[ MINE <pattern_form> ]
[ MWITH TIME <Interval/Instant> ]
[ MWHERE <interval-relation-expression> ]

The TIME_SLICE clause retrieves the data which are valid during the interval specified in the TIME_SLICE option. The WHEN clause is used to call the interval operators. The WITH clause is used to invoke the temporal reasoning tasks using the FORECAST, ANALYSE or PLAN operators. The LEARN option may be used with any of these three operators if the user wishes to add new knowledge to the rule base which are acquired from previous queries. The pattern_form gives the form of patterns, which the user would like to discover. The temporal features of the expected patterns can be expressed by MWITH and MWHERE clauses, describing the periodicity that mining users are interested in with or without a specific time framework. When any of the extended options are specified by a user, the query processor requests the rule manager to fire suitable rules. The SELECT statement can be used for querying the physical database, data dictionary and the rule base. For example, the SELECT statement,

```
SELECT * FROM EMP.RUL
```

retrieves and displays the rules that are stored in the rule base for the employee management system.

4.1.5.1 Prediction

Given a description of the world over some period of time and the set of rules governing change, predicting the world at some future time is
called prediction. In this system, prediction is performed non-monotonically on insert, delete, select and update operations. For doing so, the grammar provides an option called FOR FORECAST. The action for this option calls the curve fitting function that has been written using the method of least square for a polynomial degree n. During run time, the system computes and find the degree of the polynomial. For the given degree, the system fits a suitable curve and applies suitable rules for forecasting.

For example, consider the relations given in Figure 4.1

| EMP(EMPNO, ENAME, EDESIGN, EADDRESS, DOB) |
| SALARY(EMPNO, SALR, STIME, ETIME) |
| DEPT(EMPNO, DEPT_NAME, STIME, ETIME, SYSTIME) |
| MANAGER(EMPNO, MGR, STIME, ETIME) |

**Figure 4.1 Relational Schema**

To find the employee names whose salaries are equivalent in the same period of time and to forecast their salaries after 4 years, the query

SELECT ENAME FROM SALARY A, SALARY B
WHERE A.EMPNO = B.EMPNO AND A.SALR = B.SALR AND
WHEN A.INTERVAL EQUIVALENT B.INTERVAL
WITH FORECAST FOR SALR ON 4

displays the names of the salaries of equivalent employees using the rule.

Whenever the query processor detects a FORECAST option, it invokes the rule manager for prediction. The rule manager fires suitable rules and performs the prediction using the time series analysis method. If the user gives a query with the non-monotonic option for insert/delete/update upto a
specific time limit, the rule manager fires IF..THEN rules to perform deductive retrieval non-monotonically where the rules are active upto the time specified in the query.

### 4.1.5.2 Explanation

Given a description of the world over some period of time and the rules governing changes, explanation produces a description of the world at some earlier time that accounts for the world being the way it is at the later time. Explanation of the past in this way can also be called as counter-historical. Explanation is performed by the use of curve fitting as well as rules. Explanation can be performed by rolling back the database to a previous state specified by the query and by applying prediction on this state where the current time is substituted for the future time value. A set of rules are provided by the rule manager that are accountable for the conversion of the past state into the present state.

For example, to perform an explanation for the query “What would have been the designation of the employee 102 if he was promoted 4 years before?”, the user uses the SELECT statement:

```sql
SELECT EMPNAME, EDESIGN
FROM EMP, DEPT
WHERE EMP.EMPNO = 102 AND EMP.EMPNO = DEPT.EMPNO
WITH ANALYZE FOR EDESIGN ON 4
```

During the execution of this query, the query processor transfers the control to the inference engine component of the rule manager which is
responsible for providing the explanation by selecting and firing the necessary rules.

4.1.5.3 Planning

Given a description of some desired state of the world over some period of time, and the rules governing change, planning produces a sequence of actions that would result in a world fitting that description. Planning can be derived from historical data and current data by the rule manager using the activity monitoring component of the rule manager which uses the simulation techniques and rules for providing the various cost estimations. Planning option is used for decision making in the temporal reasoning subsystem. For example, to answer the query “Find the employee who immediately succeeded manager “VIVEK” and the time of occurrence of this event. Also find the possible managers for the next 4 years”.

The user can give the ETMSQL command

```
SELECT B.MGR, B.STIME
FROM MANAGER A, MANAGER B
WHERE A.EMPNO = B.EMPNO AND A.MGR = "VIVEK"
WHEN B.INTERVAL FOLLOWS A.INTERVAL
WITH PLAN FOR MGR ON 4
```

This query is executed to give a set of names who can be selected as manager. These sets of names are prepared on encountering the PLAN option and by the invocation of the specific rules.
4.1.5.4 Learning

Given a description of the world at different times, learning produces the rules governing change which account for the observed regularities in the world. The rule manager in coordination with the temporal database manager takes care of learning new rules during each phase of the non-monotonic temporal reasoning if the query consists of a “WITH LEARN” option.

4.1.6 ROLLBACK STATEMENT

The syntax of the ROLLBACK statement is:

ROLLBACK tablename, NUM;

The Rollback statement takes the required data from the history table under a time_slice condition. First, the time intervals are checked to retrieve the values from the history table.

For example, the ROLLBACK statement

ROLLBACK EMP, 20070202;

selects data from the base table and the history table for the time 20070202.

4.2 IMPLEMENTATION OF RULE SYSTEMS

Rule systems provide rich representation schemes, deductive search techniques and powerful reasoning methods. Applications developed using the integrated temporal database and rule systems can tackle the past, present
and future data values effectively using extended relational database constructs in coordination with production rules. In this work, a rule system which consists four types of rules namely active IF..THEN rules, passive IF..THEN rules, active cascading IF..THEN rules and passive cascading IF..THEN rules has been proposed and implemented. The salient features of the rule system are the ability to handle incomplete temporal data, deductive temporal reasoning, flexible addition and deletion of rules, specification and triggering of rules on multiple tables. The rule system developed in this work for temporal reasoning enables the query processor to check for integrity, security and consistency.

4.2.1 Creation of Rule

ETMSQL provides commands for the creation of rules. The syntax of the create_rule command is:

\[
\text{Rule\_defn :: CREATE\_RULE rule\_name} \\
\text{ON \{ALWAYS/DURING interval/etmsql\_stmt/event\}} \\
\text{WITH PRIORITY pr\_op;} \\
\text{AS IF (Conditional expression)} \\
\text{THEN Action;}
\]

All the four types of rules are created using create_rule command. This command creates rules and stores them in the rule base. Priority can be assigned for every rule using this command. A rule is declared as an active rule if it uses the ALWAYS option and these rules are used as alerters. The rules which are triggered by a transaction execution are called as a cascading rule if their action part triggers the execution of another rule. In order to provide sample commands for creating and attaching rules, consider the following database schema for a hospital information system:
DOCTOR (DNAME, AGE, SALARY, SPECIALIZATION, DNO, JOBNO)
PATIENT (WARDNO, PNAME, CONDITION, TEMPERATURE, DNO)
SURGERY (JOB, DNAME, PNAME)

Now a rule can be created on the doctor table as follows:

CREATE_RULE INTEG_RULE ON DOCTOR
FOR ALWAYS WITH PRIORITY 2
AS IF TEMPERATURE > 100 THEN
SELECT DNAME, PNAME, SPECIALIZATION
FROM DOCTOR, PATIENT
WHERE DOCTOR.DNO = PATIENT.DNO

This rule can be attached to the patient table using the following command.

ATTACH RULE INTEG_RULE
TO PATIENT

A cascading rule can be created as follows:
CREATE RULE CASC-RULE ON DOCTOR
FOR UPDATE DOCTOR
SET SALARY = SALAR * 1.1
WHERE AGE > 40
AS IF EXISTS ( SELECT * FROM PATIENT WHERE CUR-STATUS < PRE-STATUS )
THEN
SELECT * FROM PATIENT WITH FORECAST AND LEARN
FOR STIME = STIME + 5;
4.2.2 Operation on Rules

A rule created using the create_rule command can be attached to one or more objects using the attach_rule command. The common integrity constraints and validation procedures are written as rules and they are attached with applications whenever needed. Two special operators namely enable and disable are defined on temporal database rules to change the temporal constraints enforced on rules so that these operations can selectively enable and disable a rule for specified interval of time. If a rule r is disabled for an object e at an interval [a,b], this means that r will be evaluated for e during the interval [a,b]. Rules in temporal database follows the immediate coupling mode which means that rules are fired immediately before the transaction or immediately after the transaction.

4.2.3 Rule Processing

The rule processing takes place in two phases namely rule matching and rule execution.

4.2.3.1 Rule Matching

The Temporal database system proposed in this work uses the Extended Rete’s algorithm presented below for rule matching.

Extended Rete’s Algorithm

The extensions provided to the Rete’s algorithm (Forgy 1982) are used to tackle the temporal issues and also to allow the firing of four types of rules. The use of Rete’s algorithm allows the system to validate variable bindings before the query is processed. The construction of discrimination
networks makes the performance uniform throughout a transaction. Moreover, temporal constraints are also checked before the query is scheduled for execution. Hence, this temporal database system uses the Extended Rete’s algorithm given in Figure 4.2. This Extended Rete’s algorithm consists of two phases namely Construction of a tree and usage.

I Construction of a tree

1. For each antecedent pattern that appears in the rule set, create a SELECT operation for an interval \([a,b]\) that examines tuples \(t_1, t_2, \ldots, t_n\) from both base table and history table.

2. For each rule,
   (a) For each antecedent for the specified time,
       Create an alpha node and attach it to the corresponding temporally constrained SELECT operation, already created.

3. For each alpha node, except the first
   (a) Create a beta node.
   (b) If the beta node is first node,
       Attach it to the first and second alpha nodes.
       Apply TIME SLICE and temporal constraints on these nodes.

   Else

   Attach the beta node to the corresponding alpha node and to the previous beta node, and constrain the beta node using the intervals \([t_1, t_2]\).
4. Attach a projection operation which will project the non-temporal attributes with elimination of duplicate attributes along with the temporal attributes to the final beta node.

5. Perform a Join operation based on temporal attributes.

II Usage

1. For each tuple,
   
   (a) Filter the tuple through the temporal SELECT operation.
   
   (b) Pass the tuple along the Rete to the appropriate alpha nodes.

2. For each alpha node receiving a tuple,
   
   (a) Use the temporal projection operation to isolate the appropriate variable bindings.
   
   (b) Pass these new variable bindings, if any along the Rete to the next beta node or to the final temporal project operation.

3. For each rule, use the temporal project operation to isolate the variable bindings needed to instantiate the consequent.

4. Apply temporal constraints using interval operators.

**Figure 4.2 Extended Rete’s Algorithm**
In this work, the extended Rete’s network is used for rule matching in this temporal database system due to several reasons.

- This extended Rete’s network provides a semantics which performs temporal reasoning without involving conflict-resolution procedures.
- Extended Rete’s approach enables to perform the temporal relational operations incrementally. In this approach, as each new insertion is made, it performs a select operation to reduce both the total amount of computations and the storage space by allowing only a few selected tuples.

### 4.2.3.2 Rule Execution

The rule execution process consists of two phases namely rule selection and rule firing. In rule selection, rules are selected on the basis of the rule selection algorithm given in Figure 4.3.

#### Rule Selection Algorithm

Let \( R_i = \{ R_1, R_2, \ldots, R_n \} \) be the rules present in the rule base. Then select the rules using the following steps:

1. Sort the rules in descending order of priority numbers and temporal ordering and keep them in a queue.
2. Identify the rules with same priority numbers and order them temporally based on start time.
3. Assign priority numbers to such rules based on valid times and keep them in descending order of this new priority.
4. Repeat until \( \{R_i\} = \text{empty} \)
   
   (a) Select the first rule (front) from the queue.
   
   (b) Send the selected rule to the scheduler along with the result of the matching procedure.
   
   (c) Check for temporal constraints.
   
   (d) Delete the first rule from the queue.

**Figure 4.3  Rule Selection Algorithm**

In rule firing, the statements in the THEN part of the rule are given to the query processor by the scheduler. The query processor rewrites the queries to include the temporal constraints given by the rules and the rewritten query is executed. There are many reasons to choose the rule manager based query processing approach for effective query processing because of the following:

- A user who wishes to interact with the database system enforcing higher level of integrity, security and consistency can achieve it through the rule manager.
- There are a variety of applications where the end user wants to add and delete the rules based on time, to perform non-monotonic temporal reasoning under the open world assumption.
- Temporal Rules allow the capturing of temporal dependencies among intervals by facilitating the query processor to choose either a forward chaining control flow or backward chaining control flow to tackle the recursive or cascading constraints occurring from queries.
Query processing with rule firing approach takes place in two phases namely rule ordering and query execution with rules.

**Rule ordering**

The rule manager schedules the rules by ordering them and then sending them to the query processor. In the temporal database, there is an implicit temporal ordering between rules built from two kinds of ordering called chain temporal ordering and action temporal ordering.

The chain temporal ordering expresses that a rule precedes another one if the result of first rule is needed by the second rule. This ordering is reflexive, anti symmetric and transitive. It is defined as follows:

Let \( r_1 \) and \( r_2 \) be two rules defined over an interval \([a,b]\). Then \( r_1 \) is said to precede \( r_2 \) in a chain temporal ordering \((ct)\), denoted by \([r_1 \leq r_2]_{ct}\), iff there exists relations \( T_1 \) in \( r_1 \) and \( T_2 \) in \( r_2 \) such that \( T_1 \) defines \( T_2 \) denoted by equation 4.1.

\[
t_2 = f(t_1)
\]  

(4.1)

The action temporal ordering expresses that a rule precedes another rule if the former rule performs a transaction in one relation at a given interval of time and the latter rule performs a transaction in the same or different relation during the same period of time. Now the definition of action temporal ordering is given as follows:

Let \( r_1 \) and \( r_2 \) be two rules defined over an interval \([a,b]\). Then \( r_1 \) is said to precede \( r_2 \) in action temporal ordering \((at)\), denoted by \([r_1 \leq r_2]_{at}\), iff there exists a relation \( T_1 \) such that \( T_1 \) \((tr_1)\) is a transaction processed as an
action of r1 and T1 (tr2) is a transaction as an action part of r2 and there is no relation T2 such that T1 (tr1) is an action for r1 and T2 (tr2) is an action for r2.

Using these definitions of temporal rule ordering, the general definition of implicit ordering is given as:

Let r1 and r2 be two rules defined for an interval [a,b] specified above. Then r1 precedes r2, denoted by r1 ≤ r2, iff either

(i) [r1 ≤ r2] at and [r1 is not ≤ r2]ct
(ii) [r1 ≤ r2]ct and [r1 is not ≤ r2]at.

This ordering is reflexive, symmetric and transitive.

Query Execution

The execution of rules are performed by the query processor using the algorithm given in Figure 4.4.

Query Execution algorithm :

user_request = true;
do while(user_request = true)
begin
begin transaction
Initiate Temporal Query Processing;
initialize temporal read_rules;
while (there are temporal rules in the ordered queue ) do
begin
select the first temporal rule;
apply the selected rule to query and perform query rewrite;
delete the temporal rule from query;
end;
perform transaction operation;
terminate Temporal Query Processing;
end transaction;
Check for further request;
end

**Figure 4.4 Query Execution algorithm**

Each application of a rule in this algorithm makes a transition. The transition effects of a cascading rule transforms the initial state $S_0$ of a database to a final state $S_n$ in the following way:

Let $T_1$ be an externally generated transition with effect $E_1$ to transform the initial state $S_0$ to the final state $S_1$ by the application of a rule $R_1$ during an interval $[a,b]$.

![Transition Diagram](image)

If $R_1$’s condition initiates another rule $R_2$ and if the transition effect $T_2$ is generated with effect $E_2$, then a new state $S_2$ is produced as follows:

![Transition Diagram](image)

When $n$ rules are applied in deduction process, the transition produce the state $S_n$, called the final state as follows:
The query processor maintains the binding between the tuple variables appearing in the rule condition and occurrences of those tuple variables in the rule action. Commercial database systems restrict the number of rules on any object. On the other hand, this temporal database management system does not restrict the number of rules on any object. Rules can be specified during database creation time (static) or rules can be created and attached dynamically as and when it is necessary (dynamic). Hence, rules can be added or deleted dynamically using ETMSQL in this temporal database. Static rules are used for maintaining integrity while dynamic rules are used for the various database services such as security, view maintenance in the form of materialized views and version maintenance where dynamic schema definitions are used. Other services provided by this temporal database rules are audit trial, trim space, persistence clipping, temporal reasoning with null handling, data verification and data validation.