RESEARCH PAPER | Information Technology | Volume : 5 | Issue : 6 | June 2015 | ISSN - 2249-555X

Adaptation of Temporal Relational Database Models for Node.js

KEYWORDS: temporal database model, temporal databases, Node.js, asynchronous programming

DimitarPilev

Department of Informatics, University of Chemical Technology and Metallurgy,1756, bul. Cl.Ohridski, 8, Sofia,Bulgaria

ABSTRACT *There are multiple areas which require monitoring the data modification in the course of time. In this case, the databases have to store not only the current values of the data processed, but also their previous, even future, values. The purpose of this paper is to adapt a temporal relational database model for Node.js platform. A data base updating functionality is implemented. The asynchronous execution of the requests in Node.js allows quick and effective processing of bulk information stored in the temporal database.*

Introduction

There are multiple areas which require monitoring the data modification in the course of time. In this case, the databases (DB) have to store not only the current values of the data processed, but also their previous, even future, values. These are the data necessary for arrangement and execution of competitions, recruiting employees, implementation of public procurements, project management, etc. The traditional DB models may store and process only one current state of the subject field under modelling. Usually, these states are current states and when they need to be modified, their old values are deleted. The temporal database models take into account the database changes in time. A number of temporal database relational models are known [1], [2], [3].

The DBMS existing so far did not allow for temporal data processing. Considering the exceptional relevancy of the issue, the commercial DBMS, such as Oracle and Teradata, have already published [4], [5] new specifications for temporally supported DBMS. The PostgreSQL[6] DBMS has defined temporal data of the interval type and functions for their processing. An additional superficial layer within the MySQL DBMS has been developed for realization of a temporal support for the effective temporal model (ETM) in [7] which includes the operations of adding, deletion and modification of data.

The purpose of this paper is to adapt ETM [7], [8] for Node.js platform [9]. Node.js is a platform built on the basis of Google V8 JavaScript [10] for easy construction of quick and scalable network applications. Node.js uses "event-driven" asynchronous input/output operational model using only one process which makes it easy and effective in real time applications.

Effective Temporal Database Model

ETM can report the time period in which the data of the respective modeled field is simultaneously valid and recorded in the DB.

The effective time is the time period, during which the modeled field data is simultaneously valid and recorded in the DB. The current recording time is set as a beginning of the effective time period. The time period end is a future moment, which defines the end of the data validity.

Effective Temporal Model (ETM) is defined as a homogenous data model using the effective time when marking the data validity period in the tuple.

In the ETM all tuple data is marked with one and the same time. The relation is always in first normal form (1NF). The relation scheme is formed as follows. Two compulsory attributes DATE type: Es (beginning of the effective time period) and Ee (end of the effective time period) are added to the standard non temporal attributes. The effective time period is characterized with a closed lower and opened upper bound.

The relation scheme in the ETM is as follows: $R=(A_1,A_2,\ldots,A_n,E_s,E_e)$

Effective relation is the relation, which uses a period of effective time for tuple data marking. In the effective relation only valid in the recording moment facts can be added. This means that facts which were or are going to be valid in respect to the current moment (CT) cannot be added in the relation. The relation can be considered as composed of two tuple types – active and inactive.

Active are the tuples for which is valid $CT \in t_e$ (figure1a). All active tuples define the active relation state. The inactive tuples store data with expired validity period (figure1b).

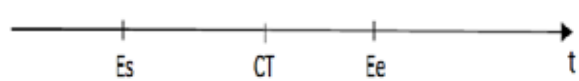

Fig.1a Activetuple, Es ≤ CT ∧ CT < Ee

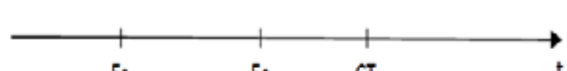

Fig. 1b Inactive tuple, CT > Ee

Data inserting in the effective relation is executed when there is need to record in the DB unrecorded facts ($a_1,\ldots,a_n$) with limited validity period. From the moment of

Volume : 5 | Issue : 6 | June 2015 | ISSN - 2249-555X

recording in the DB onwards the effective time coincides with the valid one. This indicates that the facts are both valid in the modeled reality and recorded in the DB. After the data recording a new updated relation version is created.

Algorithm for adding data in the effective relation
The values of the non temporal attributes $(a_1,\ldots,a_n)$, $t_e=[E_s,E_e)$ recorded in the DB are associated with the period of time t_e. The new time period is indicated with t_e'. It marks the same non temporal attributes values $(a_1,\ldots,a_n)$, which have to be recorded in the DB, thus $t_e'=[E_s',E_e')$.

The relation r, which is being updated, the non temporal attributes values $(a_1,\ldots,a_n)$ as well as their end of marking effective time period are set as entrance parameters of the effective relation insert algorithm.

insertET(r,$(a_1,a_2,\ldots,a_n)$,E_e'):

CT ←current_time;

e_m←0;

if$CT<E_e'$

for each x∈r

if$x[A]=(a_1,a_2,\ldots,a_n)$ *and* $x[E_e]\geq CT$

$e_m \leftarrow x[E_e]$;

break;

if $e_m = 0$

$y[A]\leftarrow(a_1,a_2,\ldots,a_n)$;

$y[E_s]\leftarrow CT$;

$y[E_e]\leftarrow E_e'$;

r←r {y};

else *if* $e_m=CT$

for each x∈r

if $x[A]=(a_1,a_2,\ldots,a_n)$ *and* $x[E_e]=CT$

$x[E_e]\leftarrow E_e'$;

break;

return r;
This algorithm can be described as follows

1. If the set E_e' value satisfies the $CT<E_e'$ model requirements then p.2 comes into action. Otherwise, the algorithm execution is discontinued and p.6 comes into action.

2. The value of the attribute E_e used to mark the attributes $(a_1,\ldots,a_n)$ values of active tuple in the relation r should be found. P.3 comes into action.

3. If the attributes $(a_1,\ldots,a_n)$ values are not part of the r's active state and the end of the marking period E_e does not coincide with the current time, a tuple with attributes $(a_1,\ldots,a_n)$ should be added in the relation r. P. 5 comes into action. Otherwise, p. 4 comes into action.

4. If the attributes $(a_1,\ldots,a_n)$ values are not part of the relation's active state and the end of the irmarking period E_e coincides with the current time, then the effective time is updated and for its end E_e' is set.

5. End of the algorithm.

Algorithm for effective relation data deleting
The effective relation r tuple deleting $(a_1,\ldots,a_n)$ is executed as follows. The current moment at the deleting request execution is assigned as an end for the effective time period. Only tuples, which belong to the relation r active state, could be deleted.

The relation r and the non temporal attributes values $(a_1,\ldots,a_n)$ are set as entrance parameters of the effective relation delete algorithm.

delete*ET*(r,$(a_1,\ldots,a_n)$):

CT ←current_time;

for each x∈r

if$x[A]=(a_1,a_2,\ldots,a_n)$ and x[Ee]>CT

$x[E_e]\leftarrow$CT;

break;

return r;
Algorithm for effective relation data updating
The existing tuple modification is executed by the operation update. It is defined as consecutive execution of the operations delete and inserts.

update*ET*(r,$(a_1,a_2,\ldots,a_n)$,$(a_1',a_2',\ldots,a_n')$,E_e'):

r ←delete*ET*(r,$(a_1,\ldots,a_n)$);

r ←insert*ET*(r,$(a_1',a_2',\ldots,a_n')$,E_e');

return r;
ETM considerably reduces the volume of the data stored and, at the same time, is not related to the loss of useful information. However, on the other hand, this model does not store information about this portion of the valid time during which the facts are not written in the DB, as well as about the modification of the end of the effective period Ee.

Adaptation of ETM for a platform on Node.js
Node-mysql[11] driver is used as a basis in the development of Node.js temporal support for MySQL database. New capabilities are added to the functionality provided by the driver for adding, deletion or modification of data, marked with effective time. For the implementation of the requests, related to data updating in effective relation the function queryET(options, callback) is used. The function requires two parameters - options and callback. The second parameter is optional and represents a function which is executed when the database server returns the result of the request made. The first parameter (options) represents a JavaScript object, containing information on the table being updated (table);

Volume : 5 | Issue : 6 | June 2015 | ISSN - 2249-555X

the type of the request – adding, modification or updating of data (operation); a list of the attributes of the request, accompanied by their values (values); the end of the time period marking the data in the tuple (periodEnd).

Figure 2 illustrates the way to add data in effective relation 'student', having the following relational scheme:

student(name, family, fnum, specialty, studygroups, course, ts, te)

Following connection to the MySQL database server the functionqueryET() is requested. Upon adding data, table, operation, values and periodEnd are given as mandatory properties of the parameter options.

```
conn.queryET({
table: 'student',
operation: 'insert',
values: {
name: 'Dan',
family: 'Lambert',
fnum: 'AU0093',
specialty: 'Automation,,
studygroup:23,
course: 2
        },
periodEnd:'2015-09-15'
},function(err, res){
//…
});
```

Fig.2 Adding data to relation 'student'

Upon deletion of a certain tuple of the effective relation, no input of the end of the effective time period is requested marking the tuple in the relation (Fig.3). In this case, even if such time is given through the attribute periodEnd of the object options, it will not be taken into account. Upon deletion of a tuple marked by effective time, only the time current as of the moment of submission of the request for deletion is taken into account.

```
conn.queryET({
table: 'student',
operation: 'delete',
values: {
name: 'Robin',
family: 'Owen',
fnum: 'MD0250',
specialty: 'Informatics',
studygroup:3,
course: 1
        }
},function(err, res){
// …
});
```

Fig.3 Deletion of data in relation 'student'

For modification of a certain tuple of the effective relation, the new values of the attributes should also be given through property newValues of the parameter options (Fig.4). If no such property is available, an error will be generated as a result of the execution of the request for modification.

```
var options = {
table: 'student',
operation: 'update',
values: {
name: 'Dan',
family: 'Lambert',
fnum: 'AU0093',
specialty: 'Automation',
studygroup:23,
course: 2
        },
newValues: {
name: 'Dan',
family: 'Lambert',
fnum: 'BT2564',
specialty: 'Biotechnology',
studygroup:15,
course: 3
        },
periodEnd:'2015-09-15'
};
conn.queryET(options, function(err, res){
```

Volume : 5 | Issue : 6 | June 2015 | ISSN - 2249-555X

```
// …

});
```

Fig.4 Modification of data in relation 'student'

In case the parameters of the temporal request do not comply with the type of the operation performed (adding, deletion or modification of data), an error is generated and the execution of the request is terminated.

Conclusion

This paper lays out algorithms for updating data of relational database using effective temporal model. The algorithms are adapted to asynchronous software model on the platform Node.js. The media allows quick and effective processing of bulk information stored in temporal databases. This is achieved through the asynchronous processing of the requests made to the databases. A new functionality is added to the standard Node.js driver used to connect to MySQL database, allowing adding, deletion and modification of data marked with effective time.

REFERENCE [1]. Christian Jensen, Richard Snodgrass,Temporal Database Entries for the Springer Encyclopedia of Database Systems. A Timecenter Technical Report, May 22, 2008. | [2]. Christian Jensen, Temporal Database Management. 2000. | [3]. C. j. Date, Nikos A. Lorentzos, Temporal data and the ralational model. 2003. | [4]. Database, O., Workspace Manager Developer's Guide. September 2010. | [5]. Database, T., Temporal Table Support. Teradata Labs, 2012 | [6]. Postgres, T. Maintaining and querying time data in PostgreSQL. Available from: http://temporal.projects.pgfoundry.org/tutorial.html. | [7]. DimitarPilev, AnetaGeorgieva, Effective Time Temporal Database Model, International Journal on Information Technologies and Security, 2012. N2: 33-46, ISSN 1313-8251 | [8]. D. Pilev, An Implementation of Effective Time Temporal Database Model in a Web-Based Information System, Revue électroniqueinternationale pour la science et la technologie, Numero 10, 2015, ISSN 1313-8871 | [9]. Node.js v0.12.0 Manual & Documentation, NodeJS, availeble: https://nodejs.org/api/modules.html | [10]. Google V8, V8 JavaScript Engine, available: https://code.google.com /p/v8/ | [11]. A pure node.js JavaScript Client implementing the MySql protocol, available: https://github.com/felixge/node-mysql/ |