TIDAQL: A Query Language Enabling On-line Analytical Processing of Time Interval Data

Philipp Meisen, Diane Keng, Tobias Meisen, Marco Recchioni and Sabina Jeschke

Abstract Nowadays, time interval data is ubiquitous. The requirement of analyzing such data using known techniques like on-line analytical processing arises more and more frequently. Nevertheless, the usage of approved multidimensional models and established systems is not sufficient, because of modeling, querying and processing limitations. Even though recent research and requests from various types of industry indicate that the handling and analyzing of time interval data is an important task, a definition of a query language to enable on-line analytical processing and a suitable implementation are, to the best of our knowledge, neither introduced nor realized. In this paper, we present a query language based on requirements stated by business analysts from different domains that enables the analysis of time interval data in an on-line analytical manner. In addition, we introduce our query processing, established using a bitmap-based implementation. Finally, we present a performance analysis and discuss the language, the processing as well as the results critically.

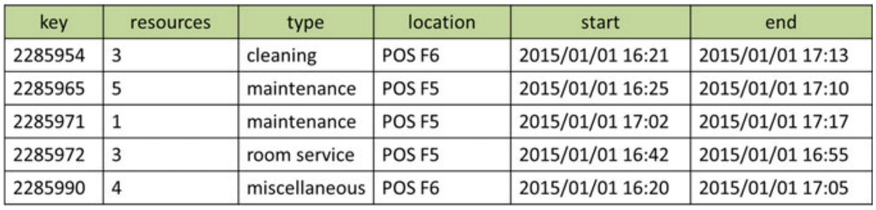
Keywords Time Interval Data · Query Language · On-Line Analytical Processing · Distributed Query Processing

1 Introduction

Nowadays, time interval data is recorded, collected and generated in various situations and different areas. Some examples are the resource utilization in production environments, deployment of personnel in service sectors, or courses of diseases in healthcare. Thereby, time interval data is used to represent observations, utilizations or measures over a period of time. Put in simple terms, time interval data is defined by two time values (i.e. start and end), as well as descriptive values associated to the interval: like labels, numbers, or more complex data structures. Figure 1 illustrates a sample database of five records.

P. Meisen (✉) · D. Keng · T. Meisen · M. Recchioni · S. Jeschke
IMA/ZLW & IfU, RWTH Aachen University,
Dennewartstr. 27, 52068 Aachen, Germany
e-mail: philipp.meisen@ima-zlw-ifu.rwth-aachen.de

Originally published in “17th International Conference on Enterprise Information Systems (ICEIS 2015)”, © 2015. Reprint by Springer International Publishing AG 2016, DOI 10.1007/978-3-319-46916-4_32



key	resources	type	location	start	end
2285954	3	cleaning	POS F6	2015/01/01 16:21	2015/01/01 17:13
2285965	5	maintenance	POS F5	2015/01/01 16:25	2015/01/01 17:10
2285971	1	maintenance	POS F5	2015/01/01 17:02	2015/01/01 17:17
2285972	3	room service	POS F5	2015/01/01 16:42	2015/01/01 16:55
2285990	4	miscellaneous	POS F6	2015/01/01 16:20	2015/01/01 17:05

Fig. 1 A sample time interval database with intervals defined by [start, end), an id, and three descriptive values

For several years, business intelligence and analytical tools have been used by managers and business analysts, inter alia, for data-driven decision support on a tactical and strategic level. An important technology used within this field, is on-line analytical processing (OLAP). OLAP enables the user to interact with the stored data by querying for answers. This is achieved by selecting dimensions, applying different operations to selections (e.g. roll-up, drill-down, or drill-across), or comparing results. The heart of every OLAP system is a multidimensional data model (MDM), which defines the different dimensions, hierarchies, levels, and members [1].

The need of handling and analyzing time interval data using established, reliable, and proven technologies like OLAP is desirable in this respect and an essential acceptance factor. Nevertheless, the MDM needed to model time interval data has to be based on many-to-many relationships which have been shown to lead to summarizability problems. Several solutions solving these problems on different modeling levels have been introduced over the last years, leading to increased integration effort, enormous storage needs, almost always inacceptable query performances, memory issues, and often complex multidimensional expressions [2, 3]. Additionally, these solutions are, considering real-world scenarios, only applicable to many-to-many relationships having a small cardinality which is mostly not the case when dealing with time interval data. As a result, the usage of MDM and available OLAP systems is not sufficient, even though the operations (e.g. roll-up, drill-down, slice, or dice) available through such systems are desired.

Enabling such OLAP like operations in the context of time interval data, requires the provision of extended filtering and grouping capabilities. The former is achieved by matching descriptive values against known filter criteria logically connected using operators like *and*, *or*, or *not*, as well as a support of temporal relations like *starts-with*, *during*, *overlapping*, or *within* [4]. The latter is applied by known aggregation operators like *max*, *min*, *sum*, or *count*, as well as temporal aggregation operators like *count started* or *count finished* [5].

The application of the *count* aggregation operator for time interval data is exemplified in Fig. 2. The color code identifies the different types of a time interval (e.g. cleaning, maintenance, room service, miscellaneous). Furthermore, the swim-lanes show the location. The figure illustrates the count of intervals for each type over one day across all locations (e.g. POS F5 and POS F6) using a granularity of minutes (i.e. 1,440 aggregations are calculated).

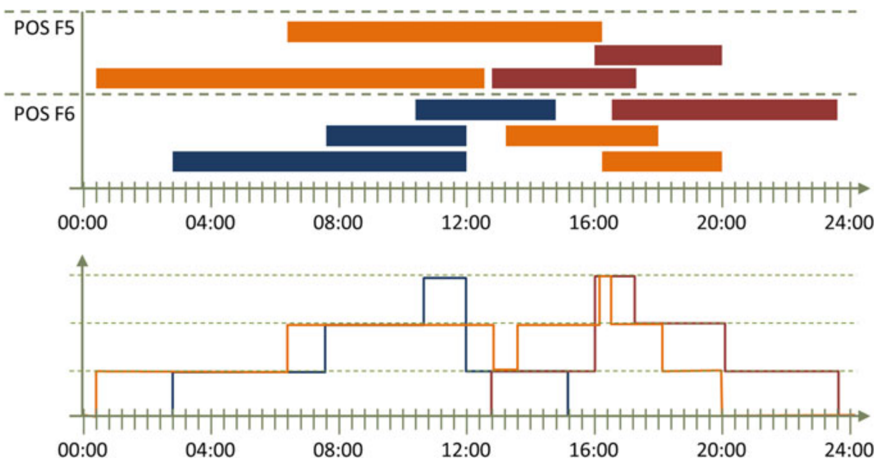


Fig. 2 On top the time interval data (10 records) shown in a Gantt-Chart, on the bottom the aggregated time-series

In this paper, we present a query language allowing to analyze time interval data in an OLAP manner. Our query language includes a data definition (DDL), a data control (DCL), and a data manipulation language (DML). The former is based on the time interval data model introduced by Meisen et al. [6], whereby the latter supports the two-step aggregation technique mentioned in Meisen et al. [5]. Furthermore, we outline our query processing which is based on a bitmap-based implementation and supports distributed computing.

This paper is organized as follows: In Section 2, we discuss related work done in the field of time interval data, in particular this section provides a concise overview of research dealing with the analyses of time interval data. We provide an overview of time interval models, discuss related work done in the field of OLAP, and present query languages. In Section 3, we introduce our query language and processing. The section presents among other things how a model is defined and loaded, how temporal operators are applied, how the two-step aggregation is supported, how groups are defined, and how filters are used. We introduce implementation issues and empirically evaluate the performance regarding the query processing in Section 4. We conclude with a summary and directions for future work in Section 5.

2 Related Work

When defining a query language, it is important to have an underlying model, defining the foundation for the language (e.g. the relational model for SQL, different interval-based models for e.g. IXSQL or TSQL2, the multidimensional model for MDX, or the graph model for Cypher). Over the last years several models have been introduced in the field of time intervals, e.g. for temporal databases [7], sequential pattern mining [8, 9], association rule mining [10], or matching [11].

Chen et al. [12] introduced the problem of mining time interval sequential patterns. The defined model is based on events used to derive time intervals, whereby a time interval is determined by the time between two successive time-points of events. The definition is based on the sequential pattern mining problem introduced by Agrawal and Srikant [13]. The model does not include any dimensional definitions, nor does it address the labeling of time intervals with descriptive values.

Papapetrou et al. [14] presented a solution for the problem of "discovering frequent arrangements of temporal intervals". An e-sequence is an ordered set of events. An event is defined by a start value, an end value and a label. Additionally, an e-sequence database is defined as a set of e sequences. The definition of an event given by Papapetrou et al. is close to the underlying definition within this paper (cf. Fig. 1). Nevertheless, facts, descriptive values, and dimensions are not considered.

Mörchen [15] introduced the TSKR model defining tones, chords, and phrases for time intervals. Roughly speaking, the tones represent the duration of intervals, the chords the temporal coincidence of tones, and the phrases represent the partial order of chords. The main purpose of the model presented by Mörchen is to overcome limitations of Allen's [4] temporal model considering robustness and ambiguousness when performing sequential pattern mining. The model neither defines dimensions, considers multiple labels, nor recognizes facts.

Summarized, models presented in the field of sequential pattern mining, association rule mining or matching do generally not define dimensions and are focused on generalized interval data, or support only non-labelled data. Thus, these models are not suitable considering OLAP of time interval data, but are a guidance to the right direction.

Within the research community of temporal databases different interval-based models have been defined [7]. The provided definitions can be categorized in weak and strong models. A weak model is one, in which the intervals are used to group time-points, whereas the intervals of the latter carry semantic meaning. Thus, a weak interval-based model is not of further interest from an analytical point of view, because it can be easily transformed into a point-based model. Nevertheless, a strong model and the involved meaning of the different operators – especially aggregation operators – are of high interest from an analytical view. Strong interval-based models presented in the field of temporal databases lack to define dimensions, but present important preliminary work.

In the field of OLAP, several systems capable of analyzing sequences of data have been introduced over the last years. Chui et al. [16] introduced S-OLAP for analyzing sequence data. Liu and Rundensteiner [17] analyzed event sequences using hierarchical patterns, enabling OLAP on data streams of time point events. Bebel et al. [18] presented an OLAP like system enabling time point-based sequential data to be analyzed. Nevertheless, the system neither support time intervals, nor temporal operators.

Recently, Koncilia et al. [19] presented I OLAP, an OLAP system to analyze interval data. They claim to be the first proposing a model for processing interval data.

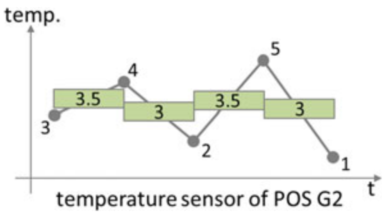


Fig. 3 Illustration of the model introduced by Koncilia et al. [19]. The intervals (rectangles) are created for each two consecutive events (dots). The facts are calculated using the average function as the *compute value function*

The definition is based on the interval definition of Chen et al. [12] which defines the intervals as the gap between sequential events. However, Koncilia et al. assume that the intervals of a specific event-type (e.g. temperature) for a set of specific descriptive values (e.g. POS G2) are non-overlapping and consecutive. Considering the sample data shown in Fig. 1, the assumption of non-overlapping intervals is not valid in general (cf. record 2,285,965 and 2,285,971). Figure 3 illustrates the model of Koncilia et al. showing five temperature events for POS G2 and the intervals determined for the events. Koncillia et al. mention the support of dimensions, hierarchies, levels, and members, but lack to specify what types of hierarchies are supported and how e.g. non-strict relations are handled.

Also recently, Meisen et al. [6] introduced the TIDAMODEL "enabling the usage of time interval data for data-driven decision support". The presented model is defined by a 5-tuple (P, Σ, τ, M, δ) in which P denotes the time interval database, Σ the set of descriptors, τ the time axis, M the set of measures, and δ the set of dimensions. The time interval database P contains the raw time interval data records and a schema definition of the contained data. The schema associates each field of the record (which might contain complex data structures) to one of the following categories: temporal, descriptive, or bulk. Each descriptor of the set Σ is defined by its values (more specific its value type), a mapping- and a fact-function. The mapping-function is used to map the descriptive values of the raw record to one or multiple descriptor values. The mapping to multiple descriptor values allows the definition of non-strict fact-dimension relationships. Additionally, the model defines the time axis to be finite and discrete, i.e. it has a start, an end, and a specified granularity (e.g. minutes). The set of dimensions δ can contain a time dimension (using a rooted plane tree for the definition of each hierarchy) and a dimension for each descriptor (using a directed acyclic graph for a hierarchy's definition). Figure 4 illustrates the modeled sample database of Fig. 1 using the TIDAMODEL. The figure shows the five intervals, as well as the values of the descriptors location (cf. swim-lane) and type (cf. legend). Dimensions are not shown. The used mapping function for all descriptors is the identity function. The used granularity for the time dimension is minutes.

Another important aspect when dealing with time interval data in the context of OLAP, is the aggregation of data and the provision of temporal aggregation operators. Kline and Snodgrass [20] introduced temporal aggregates, for which several

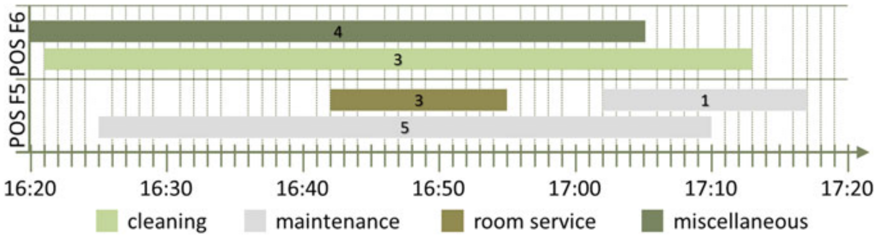


Fig. 4 Data of the sample database shown in Fig. 1 modeled using the TIDAMODEL [6]

start	end	res.	loc.
16:21	16:31	4	POS1
16:21	16:38	3	POS1
16:22	16:32	2	POS2
16:25	16:38	5	POS2

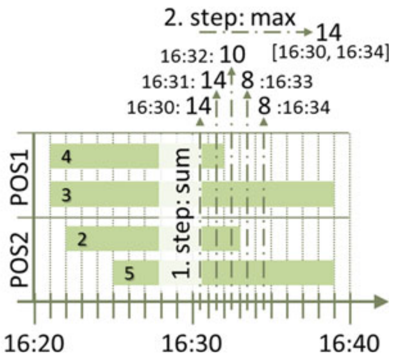


Fig. 5 Two-step aggregation technique presented by Meisen et al. [5]

enhanced algorithms were presented over the past years. Nevertheless, the solutions are focused on one specific aggregation operator (e.g. SUM), do not support multiple filter criteria, or do not consider data gaps. Koncilia et al. [19] address shortly how aggregations are performed using the introduced *compute value functions* and *fact creating functions*. Temporal operators are neither defined nor mentioned. Koncilia et al. point out that some queries need special attention when aggregating the values along time, but a more precise problem statement is not given. Meisen et al. [5] introduce a two-step aggregation technique for time interval data. The first one aggregates the facts along the intervals of a time granule and the second one aggregates the values of the first step depending on the selected hierarchy level of the time dimension. Figure 5 illustrates the two-step aggregation technique. In the illustration, the technique is used to determine the needed resources within the interval [16:30, 16:34]. Within the first step, the sum of the resources for each granule is determined and within the second step the maximum of the determined values is calculated, i.e. 14. Additionally, they introduce temporal aggregation operators like *started* or *finished count*.

The definition of a query language based on a model and operators (i.e. like aggregations), is common practice. Regarding time-series, multiple query languages and enhancements of those have been introduced [21]. In the field of temporal databases time interval-based query languages like IXSQL, TSQL2, or ATSQL have been defined [7] and within the analytical field, MDX [22] is a widely used language to query MDMs. Considering models dealing with time interval data in the context of analytics, Koncilia et al. [19] published the only work the authors are aware of that mentions a query language. Nevertheless, the query language is neither formally defined nor further introduced.

Summarized, it can be stated that recent research and requests from industry indicate that the handling of time interval data in an analytical context is an important task. Thus, a query language is required capable of covering the arising requirements. Koncilia et al. [19] and Meisen et al. [5, 6] introduced two different models useful for OLAP of time interval data. Different temporal aggregation operators, as well as standard aggregation operators, are also presented by Meisen [5]. Nevertheless, a definition of a query language useful for OLAP and an implementation of the processing are, to the best of our knowledge, not formally introduced.

3 The Tida Query Language

In this section, we introduce our time interval data analysis query language (TIDAQL). The language was designed for a specific purpose; to query time interval data from an analytical point of view. The language is based on aspects of the previously discussed TIDAMODEL. Nevertheless, the language should be applicable to any time interval database system which is capable of analyzing time interval data. Nevertheless, some adaptions might be necessary or some features might not be supported by any system.

3.1 Requirements

The requirements concerning the query language and its processing were specified during several workshops with over 70 international business analysts from different domains (i.e. aviation industry, logistics providers, service providers, as well as language and gesture research). We aligned the results of the workshop with an extended literature research. Table 1 summarizes selected results.

3.2 Data Control Language

The definition of the DCL is straight forward to the DCL known from other query languages e.g. SQL. As defined by requirement [DCL1], the language must encompass authorization features. Hence, the language contains commands like **ADD**, **DROP**, **MODIFY**, **GRANT**, **REVOKE**, **ASSIGN** and **REMOVE**. In our implementation, the execution of a DCL command always issues a direct commit, i.e. a roll back

Table 1 Summary of the requirements concerning the time interval analysis query language (selected results)

Requirement	Description
Data Control Language (DCL)	
[DCL1]: authorization aspects	It is expected that the language encompasses authorization features, e.g. user deletion, role creation, granting and revoking permissions
[DCL2]: permissions grantable on global and model level	Permissions must be grantable on a model and a global level. It is expected that the user can have the permission to add data to one model but not to another. For simplicity, it should be possible to grant or revoke several permissions at once
Data Definition Language (DDL)	
[DDL1]: loading and unloading	The language has to offer a construct to load new and unload models. The newly loaded model has to be available without any restart of the system. An unloaded model has to be unavailable after the query is processed. However, queries currently in process must still be executed
[DDL2]: non-onto, non-covering, non-strict hierarchies	Each descriptor dimension must support hierarchies which might be non-onto, non-covering, and / or non-strict [23]
[DDL3]: raster levels	A raster level is a level of the time dimension. For example: the *5-minute raster*-level defines members like [00:00, 00:05) …[23:55, 00:00). Several raster levels can form a hierarchy (e.g. 5-min → 30-min → 60-min → half-day → day)
Data Manipulation Language (DML)	
[DML1]: raw data records	The language must provide a construct to select the raw time interval data records
[DML2]: time series by time-windows	The language must support the specification of a time-window for which time-series of different measures can be retrieved
[DML3]: temporal operators	It must be possible to use temporal operators for filtering as e.g. defined by Allen [4]. Depending on the type of selection (i.e. raw records or time-series) the available temporal operators may differ
[DML4]: The two-step aggregation technique	Meisen et al. [5] present a two-step aggregation technique which has to be supported by the language. Both aggregation operators (see Fig. 5) must be specified by a query selecting time-series, no predefined measure should be necessary

(continued)

Table 1 (continued)

Requirement	Description
[DML5]: complete time series	A time-series is selected by specifying a time-window (e.g. [01.01.2015, 02.01.2015) and a level (e.g. minutes). The resulting time-series must contain a value for each member of the selected level, even if no time interval covers the specified member. The value might be N/A or null to indicate missing information
[DML6]: insert, update and delete	The language must offer constructs to insert, update and delete time interval data records
[DML7]: open, half-open, or closed intervals	The system should be capable of interpreting intervals defined as open, e.g. (0, 5), closed, e.g. [0, 5], or half-opened, e.g. (0, 5]
[DML8]: meta-information	It is desired that the language supports a construct to receive meta-information from the system, e.g. actual version, available users, or loaded models
[DML9]: bulk load	It is desired, that the language provides a construct to enable a type of bulk load, i.e. increased insert performance

is not supported. Figure 6 shows the syntax diagram of the commands. Because of simplicity, a value is not further specified and might be a permission, a username, a password, or a role.

To fulfill the [DCL2] requirement, we define a permission that consists of a scope-prefix and the permission itself. We define two permission-scopes GLOBAL and MODEL. Thus, a permission of the GLOBAL scope is defined by

```
GLOBAL.<permission>
```

(e.g. GLOBAL.manageUser). Instead, a permission of the MODEL scope is defined by

```
MODEL.<model>.<permission>
```

(e.g. MODEL.myModel.query).

For query processing, we use the Apache Shiro authentication framework (http://shiro.apache.org/). Shiro offers annotation driven access control. Thus, the permission to e.g. execute a DML query is performed by annotating the processing query method.

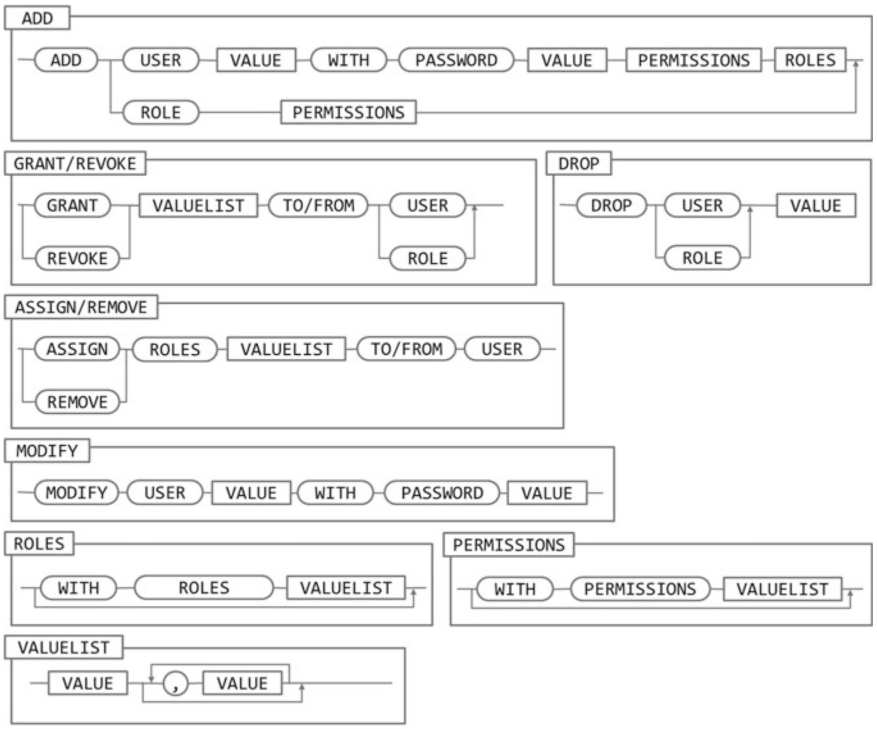


Fig. 6 Commands of the DCL query language

3.3 Data Dafinition Language

The DDL is used to define, add, or remove the models known by the system. [DDL1] requires a command within the DDL which enables the user to load or unload a model. The syntax diagram of the **LOAD** and **UNLOAD** command is shown in Fig. 7. A model can be loaded by using a model identifier already known to the system (e.g. if the model was unloaded), or by specifying a location from which the system can retrieve a model definition to be loaded. Additionally, properties can be defined (e.g. the *autoload* property can be set, to automatically load a model when the system is started). In the following subsection, we present an XML used to define a TIDAMODEL.

3.3.1 The XML TIDAMODEL Definition

As mentioned in Section 2, the TIDAMODEL is defined by a 5-tuple (P, Σ, τ, M, δ). The time interval database P contains the raw record inserted using the API or the **INSERT** command introduced later in Section 3.4.1. From a modelling perspective

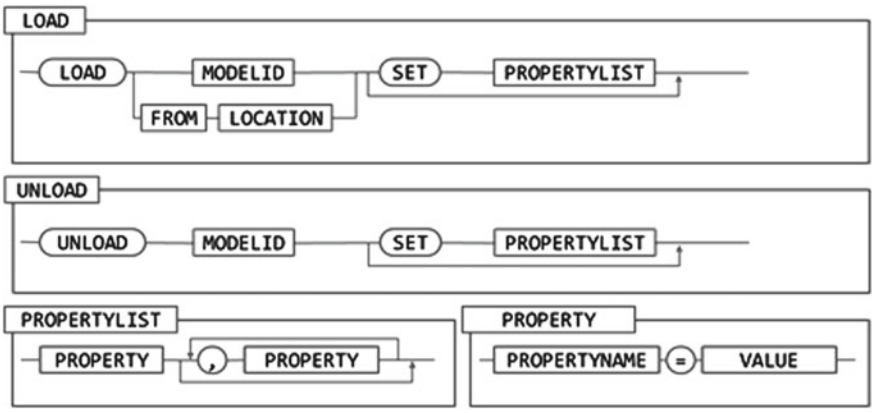


Fig. 7 Commands of the DDL query language

it is important for the system to retrieve the descriptive and temporal values from the raw record. Thus, it is essential to define the descriptors Σ and the time axis τ within the XML definition. Below, an excerpt of an XML file defining the descriptors of our sample database shown in Fig. 1 is presented:

```
<model id="myModel">
 <descriptors>
  <string id="LOC" name="location" />
  <string id="TYPE" name="type" />
  <int id="RES" null="true" />
 </descriptors>
</model>
```

The excerpts shows that a descriptor is defined by a tag specifying the type (i.e. the descriptor implementation to be used), an id-attribute, and an optional name-attribute. Additionally, it is possible to define if the descriptor allows *null* values (default) or not. To support more complex data structures (and one's own mapping functions), it is possible to specify one's own descriptor-implementations:

```
<descriptors>
 <ownImpl:list id="D4" />
</descriptors>
```

Our implementation scans the class-path automatically, looking for descriptor implementations. An added implementation must provide an XSLT file, placed into the same package and named as the concrete implementation of the descriptor-class. The XSLT file is used to create the instance of the own implementation using a Spring Bean configuration (http://spring.io/).

```
<!- File: my/own/desc/List.xslt ->
<xsl:template match="ownImpl:list">
 <xsl:call-template name="beanDesc">
  <xsl:with-param name="class">
    my.own.desc.List
  </xsl:with-param>
 </xsl:call-template>
</xsl:template>
```

The time axis of the T*IDA*M*ODEL* is defined by:

```
<model id="myModel">
 <time>
  <timeline start="20.01.1981"
            end="20.01.2061"
            granularity="MINUTE" />
 </time>
</model>
```

The time axis may also be defined using integers, i.e. [0, 1000]. Our implementation includes two default mappers applicable to map different types of temporal raw record value to a defined time axis. Nevertheless, sometimes it is necessary to use different time-mappers (e.g. if the raw data contains proprietary temporal values) which can be achieved using the same mechanism as described previously for descriptors.

Due to the explicit time semantics, the measures M defined within the TIDAMODEL are different than the ones typically known from an OLAP definition. The model defines three categories for measures, i.e. *implicit time measures*, *descriptor bound measures*, and *complex measures*. The categories determine when which data is provided during the calculation process of the measures. Our implementation offers several aggregation operators useful to specify a measure, i.e. *count*, *average*, *min*, *max*, *sum*, *mean*, *median*, or *mode*. In addition, we implemented two temporal aggregation operators *started count* and *finished count*, as suggested by Meisen et al. [5]. We introduce the definition and usage of measures in Section 3.4.2.

The TIDAMODEL also defines the set of dimensions δ. The definition differs between descriptor dimensions and a time dimension, whereby every dimension consists of hierarchies, levels, and members. It should be mentioned that, from a modelling point of view, each descriptor dimension fulfills the requirements formalized in [DDL2] and that the time dimension supports raster-levels as requested in [DDL3]. The definition of a dimension for a specific descriptor or the time dimension can be placed within the XML definition of a model using:

```
<model id="myModel">
 <dimensions>
  <dimension id="DIMLOC" descId="LOC">
   <hierarchy id="LOC">
    <level id="HOTEL">
     <member id="DREAM" rollUp="∗" />
     <member id="STAR" rollUp="∗" />
     <member id="ADV" reg="TENT"
         rollUp="∗" />
    </level>
    <level id="ROOMS">
     <member id="POSF" reg="POS F".
         rollUp="DREAM" />
     <member id="POSG" reg="POS G".
         rollUp="DREAM" />
    </level>
    <level id="STARROOMS">
     <member id="POSA" reg="POS A".
         rollUp="STAR" />
    </level>
   </hierarchy>
  </dimension>
 </dimensions>
</model>
```

Figure 8 illustrates the descriptor dimension defined by the previously shown XML excerpt. The circled nodes are leaves which are associated with de-scriptor values known by the model (using regular expressions). Additionally, it is possible to add dimensions for analytical processes to an already defined model, i.e. to use it only for a specific session or query. The used mechanism to achieve that is similar to the loading of a model and will not further be introduced.

The definition of a time dimension is straight forward to the one of a descriptor dimension. Nevertheless, we added some features in order to ease the definition. Thus, it is possible to define a hierarchy by using pre-defined levels (e.g. templates

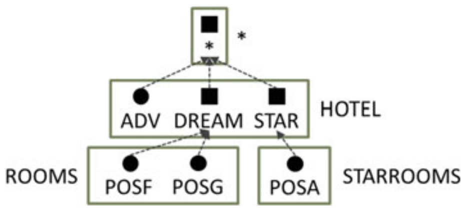


Fig. 8 Illustration of the dimension created with our web-based dimension-modeler as defined by the XML

like 5 min raster, day, or year) and by defining the level to roll up to, regarding the hierarchy. The following XML excerpt exemplifies the definition:

```
<model id="myModel">
 <dimensions>
  <timedimension id="DIMTIME">
   <hierarchy id="TIME5TOYEAR">
    <level id="YEAR" template="YEAR"
       rollUp="*" />
    <level id="DAY" template="DAY"
       rollUp="YEAR" />
    <level id="60R" template="60RASTER"
        rollUp="DAY" />
    <level id="5R" template="5RASTER"
        rollUp="60R" />
    <level id="LG" template="LOWGRAN"
        rollUp="5R" />
   </hierarchy>
  </timedimension>
 </dimensions>
</model>
```

A defined model is published to the server using the **LOAD** command. The following subsection introduces the command, focusing on the loading of a model from a specified location.

3.3.2 Processing the Load Command

The loading of a model can be triggered from different applications, drivers, or platforms. Thus, it is necessary to support different loaders to resolve a specified location. In the following, some examples illustrate the issue. When firing a **LOAD** query from a web-application, it is necessary that the model defi nition was uploaded to the server, prior to executing the query. While running on an application server, it might be required to load the model from a database instead of loading it from the file-system. Thus, we added a resource-loader which can be specified for each context of a query. Within a servlet, the loader resolves the specified location against the upload-directory, whereby our JDBC driver implementation is capable of sending a client's file to the server using the data stream of the active connection. After retrieving and validating the resource, the implementation uses a model-handler to bind and instantiate the defined model. As already mentioned, the bitmap-based implementation presented by Meisen et al. [5] is used. The implementation instantiates several indexes and bitmaps for the defined model. After the instantiation, the model is marked to be up and running by the model-handler and accepts DML queries.

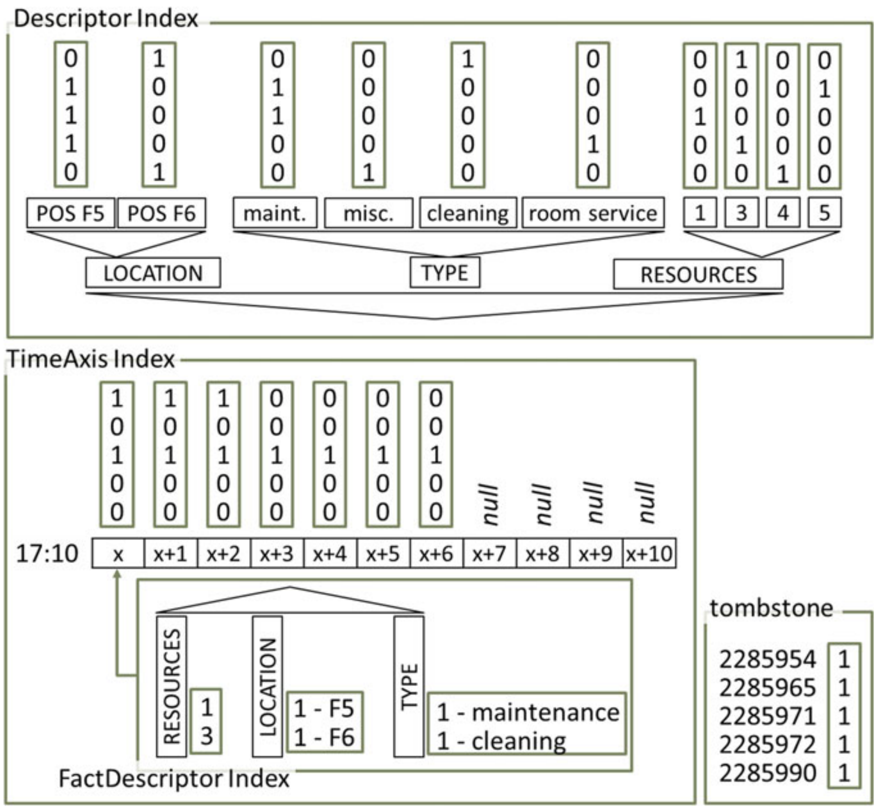


Fig. 9 Example of a loaded model [5] filled with the data shown in Fig. 1

Figure 9 exemplifies the initialized bitmap-based indexes filled with the data from the database of Fig. 1.

3.4 Data Manipulation Language

Considering the requirements, it can be stated that the DML must contain commands to **INSERT**, **UPDATE**, and **DELETE** records. In addition, it is necessary to provide **SELECT** commands to retrieve the time interval data records, as well as results retrieved from aggregation (i.e. time-series). Furthermore, a **GET** command to retrieve meta-information of the system is needed.

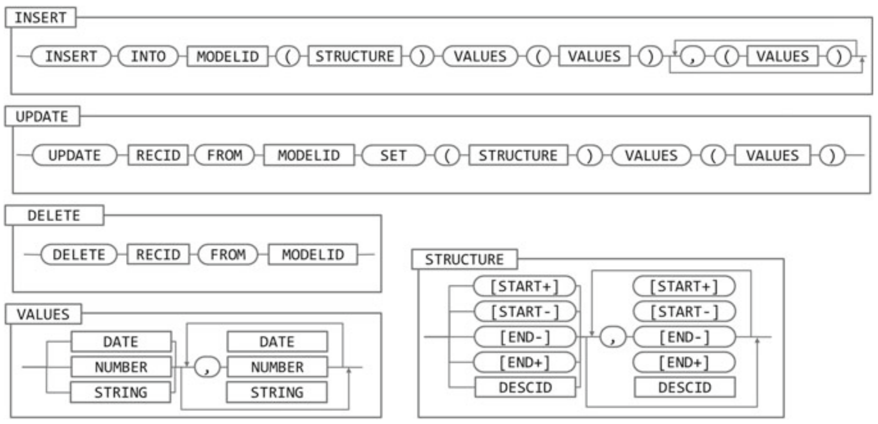


Fig. 10 Syntax diagrams of the commands **INSERT**, **UPDATE** and **DELETE**

3.4.1 INSERT, UPDATE, and DELETE

Figure 10 illustrated the three commands **INSERT**, **UPDATE**, and **DELETE** using syntax diagrams which fulfill the requirement [DML6]. The **INSERT** command adds one or several time interval data records to the system. First, it parses the structure of the data to be inserted. The query-parser validates the correctness of the structure, i.e. the structure must contain exactly one field marked as *start* and exactly one field marked as *end* even though the syntax diagram suggest differently. Additionally, the parser verifies if a descriptor (referred by its id) really exists within the model. Finally, it reads the values and invokes the processor by passing the structure, as well as the values. The processor iterates over the defined values, validates those against the defined structure, uses the mapping functions of the descriptors to receive the descriptor values, and calls the mapping function of the time-axis. The result is a so-called *processed record* which is used to update the indexes. The persistence layer of the implementation ensures that the raw record and the indexes get persisted. Finally, the tombstone bitmap is updated which ensures that the data is available within the system.

A deletion is performed by setting the tombstone bitmap for the specified id to 0. This indicates that the data of the record is not valid and thus the data will not be considered by any query processors anymore. The internally scheduled clean-up process removes the deleted records and releases the space.

An update is performed by deleting the record with the specified identifier and inserting the record as described above.

To support bulk load, as desired by [DML9], an additional statement is introduced. The statement **SET BULK TRUE** is used to enable the bulk load, whereby **SET BULK FALSE** stops the bulk loading process. When enabling the bulk load, the system waits until all currently running **INSERT**, **UPDATE**, or **DELETE** queries of other sessions are performed. New queries of that type are rejected across all sessions during the waiting and processing phase. When all queries are handled, the system responds to the bulk-enabling query and expects an insert-like statement, whereby the system directly starts to parse the incoming data stream. As soon as the structure is known, all incoming values are inserted. The indexes are only updated in memory. If and only if the memory capacity reaches a specified threshold, the persistence-layer is triggered. In this case, the current data in memory is flushed and persisted using the configured persistence-layer (e.g. using the file-system, a relational database, or any other NoSQL database). The memory is also flushed and persisted whenever a bulk load is finished.

3.4.2 SELECT Raw Records and Time-Series

The **SELECT** command is addressed by the requirements [DML1], [DML2], [DML3], [DML4], [DML5], and [DML7]. Figure 11 illustrates the select statements to select records and time-series. Because of space limitations, we removed more detailed syntax diagrams for the **LOGICAL** and **GROUP EXPRESSION**. The non-terminal **MEASURES** is specified later in this subsection when introducing the **SELECT TIMESERIES** in detail.

As illustrated, the intervals can be defined as open, half-open or closed (cf. [DML7]). The processing of the intervals is possible, thanks to the discrete time-axis used by the model. Using a discrete time-axis with a specific granularity makes it easy to determine the previous or following granule. Thus, every half-open or open interval can be transformed into a closed interval using the previous or following granule. Hence, the result of the parsing always contains a closed interval which is used during further query processing.

As illustrated in Fig. 11, the **SELECT RECORDS** statement allows to retrieve records satisfying a logical expression based on descriptor values (e.g. **LOC = "POS F5" OR (TYPE = "cleaning" AND DIMLOC.LOC.HOTEL = "DREAM")**) and/or fulfilling a temporal relation (cf. [DML3]). Our query language supports ten different temporal relations following Allen [4]: **EQUALTO**, **BEFORE**, **AFTER**, **MEETING**, **DURING**, **CONTAINING**, **STARTINGWITH**, **FINISHINGWITH**,

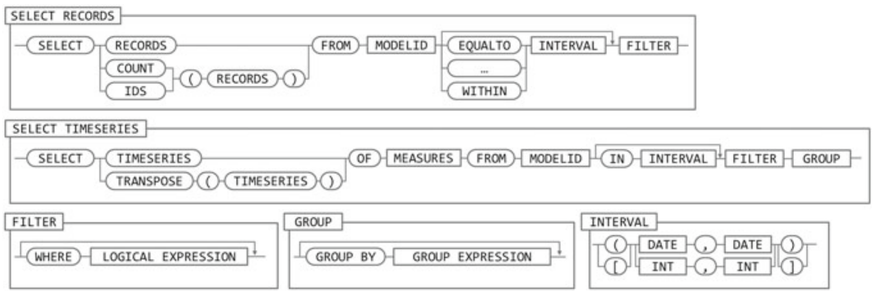


Fig. 11 Syntax diagrams of the **SELECT RECORDS** and **SELECT TIMESERIES** commands

OVERLAPPING, and **WITHIN**. The interested reader may notice that Allen introduced thirteen temporal relationships. We removed some inverse relationships (i.e. inverse of meet, overlaps, starts, and finishes). When using a temporal relation-ship within a query, the user is capable of defining one of the intervals used for comparison. Thus, the removed inverse relationships are not needed, instead the user just modifies the self-defined interval. In addition, we added the **WITHIN** relationship which is a combination of several relationships and allows an easy selection of all records within the user-defined interval (i.e. at least one time-granule is contained within the user-defined interval).

When processing a **SELECT RECORD** query, the processor initially evaluates the filter expression and retrieves a single bitmap specifying all records fulfilling the filter's logic [5]. In a second phase, the implementation determines a bitmap of records satisfying the specified temporal relationship. The two bitmaps are combined using the and-operator to retrieve the resulting records. Depending on the requested information (i.e. count, identifiers, or raw records (cf. [DML1])), the implementation creates the response using bitmap-based operations (i.e. count and identifiers) or retrieving the raw records from the persistence layer. Figure 12 depicts the evaluation of selected temporal relationships using bitmaps and the database shown in Fig. 1.

The **SELECT TIMESERIES** statement specifies a logical expression equal to the one exemplified in the **SELECT RECORDS** statement. In addition, the statement specifies a **GROUP EXPRESSION** which defines the groups to create the time-series for (e.g. **GROUP BY DIMLOC.LOC.ROOMS**). Furthermore, the measures

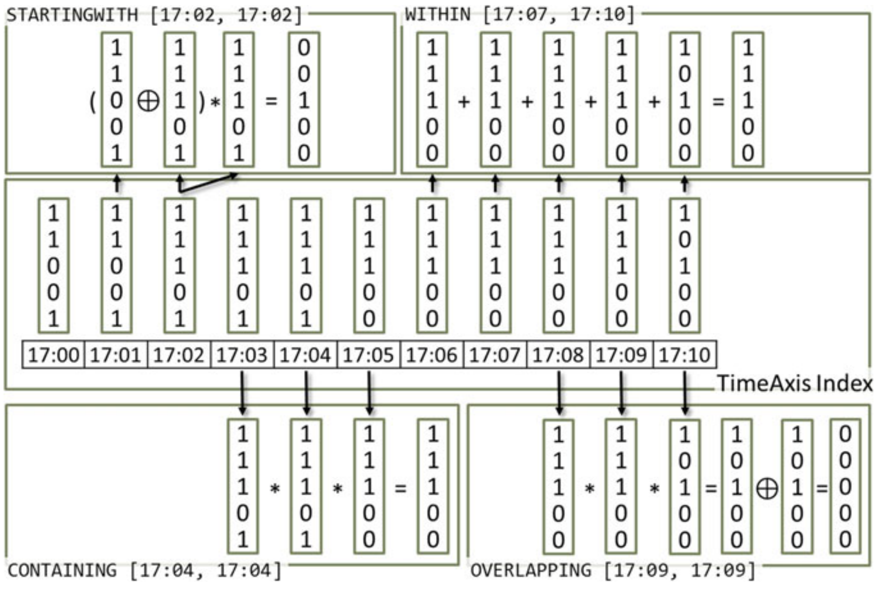


Fig. 12 Examples of the processing of temporal relationships using bitmaps (and the sample database of Fig. 1)

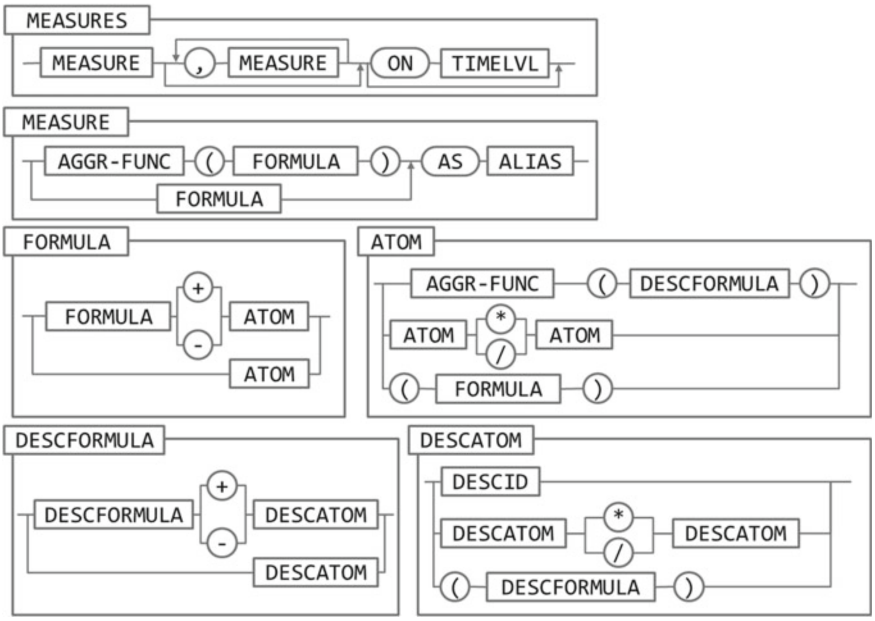


Fig. 13 Syntax diagrams of the MEASURES definition

to be calculated for the time-series and the time-window (cf. [DML2]) are specified. It is also possible to specify several comma-separated measures. Figure 13 shows the syntax used to specify measures (cf. **MEASURES** in Figure 11).

A simple (considering the measures) example of a **SELECT TIMESERIES** query is as follows:

```
SELECT TRANSPOSE(TIMESERIES)
  OF MAX(SUM(RESOURCES)) AS "needed Res"
  ON DIMTIME.TIME5TOYEAR.5RASTER
  FROM myModel
  IN [01.01.2015, 02.01.2015)
  WHERE DIMLOC.LOC.HOTEL="DREAM"
  GROUP BY TYPE
```

As required by [DML4], a measure can be defined using the two-step aggregation technique. The first aggregation (in the example SUM) is specified for a specific descriptor and the second optional aggregation function (in the example MAX) aggregates the values across the stated level of the time-dimension.

When processing the query, the system retrieves the bitmaps for the filtering and the grouping conditions. The system iterates over the bitmaps of the specified groups and the bitmaps of the granules of the selected time-window. For each iteration, the implementation combines the filter-bitmap, group-bitmap, and the time-granule-bitmap and applies the first aggregation function. The second aggregation function is applied whenever all values of a member of the specified time-level are determined by the first step (cf. Fig. 6). This processing technique ensures that for each time-granule a value is calculated, even if no interval covers the granule (cf. [DML5]).

3.4.3 GET Meta-Information

[DML8] demands the existence of a command which enables the user to retrieve meta-information, like the version of the system. This requirement is fulfilled by adding a **GET** command to the query language. A statement like **GET VERSION**, **GET USERS**, or **GET MODELS** enables the user to retrieve information provided from the system. Filtering is currently not required and thus, not supported.

4 Implementation Issues

This section introduces selected implementation aspects of the language and its query processing. First, we introduce processing implementations for the most frequently used query-type **SELECT TIMESERIES** and show performance results for the different algorithms. In addition, we present considerations of analysts using the language to analyze time interval data and address possible enhancements.

4.1 SELECT TIMESERIES Processing

In Section 3, we outlined the query processing based on the TIDAMODEL and its bitmap-based implementation (cf. Sections 3.3.2 and 3.4.2). For a detailed description of the bitmap-based implementation we refer to Meisen et al. [5]. In this section, we introduce three additional algorithms which are capable to process the most frequently used **SELECT TIMESERIES** queries, introduced in Section 3.4.2.

Prior to explaining the algorithms, it should be stated, that we did not implement any algorithm based on AGGREGATIONTREEs [20], MERGESORT, or other related aggregation algorithms defined within the research field of temporal databases. Such algorithms are optimized to handle single aggregation operators (e.g. count, sum, min, or max). Thus, the implementation would not be a generic solution usable for any query. Nevertheless, such algorithms might be useful to increase query performance for specific, often used measures. It might be reasonable to add a language feature, which allows to define a special handling (e.g. using an AGGREGATIONTREE) for a specific measure.

Next, we introduce our naive implementation. All three presented algorithm do not support queries using group by, multiple measures, nor multi-threading scenarios. To support these features, commonly used techniques (e.g. iterations and locks) could be used.

```
01 TimeSeries naive(Query q, Set r) {
02 TimeSeries ts = new TimeSeries(q);
03 // filter time def. by IN [a, b]
04 r = filter(r, q.time());
05 // filter records def. by WHERE
06 r = filter(r, q.where());
07 // it. ranges def. by IN and ON
08 for (TimeRange i : q.time()) {
09 // filter records for the range
10 r' = filter(r, i);
11 // det. measures def. by OF
12 ts.set(i, calc(i, r', q.meas());
13 }
14 return ts;
15 }
```

The algorithm filters the records of the database, which fulfill the defined criteria of the **IN** (row 04) and **WHERE** clause (row 06). Next, it calculates the measure for each defined range (row 10). The calculation of each measure depends mainly on its type (i.e. measure of lowest granularity (e.g. query #1 in Table 2), measure of a level (e.g. query #2), or two-step measure (e.g. query #3)). Because of space limitations, we state the complexity of the **calc**-method instead of presenting it. The complexity is **O(k·n)**, with k being the number of granules covered by the **TimeRange** and n being the number of records.

The other algorithms we implemented are based on INTERVALTREEs (INTTREE) as introduced by Kriegel [24]. The first one (A) – of the two INTTREE – based implementations – uses the tree to retrieve the relevant records considering

Table 2 The shortened queries used for testing

#	Query
1	**OF** COUNT(TASKTYPE **IN** [01.JAN, 01.FEB)
	WHERE WA.LOC.TYPE='Gate'
2	**OF** SUM(TASKTYPE) **ON** TIME.DEF.DAY
	IN [01.JAN, 01.FEB) **WHERE** WORKAREA='SEN13'
3	**OF** MAX(COUNT(WORKAREA)) **ON** TIME.DEF.DAY
	IN [01.JAN, 01.FEB) **WHERE** TASKTYPE='short'

the **IN**-clause (row 05 of the naive algorithm). Further, the algorithm proceeds as the naive algorithm.

The second implementation (B) differs from the first one, by created a new INTTREE for every query.

```
01 TimeSeries iTreeB(Query q, Set r) {
02 TimeSeries ts = new TimeSeries(q);
03 // filter records def. by WHERE
04 IntervalTree iTree =
05 createAndFilter(r, q.in(),
06 q.where());
07 // it. ranges def. by IN and ON
08 for (TimeRange i : q.time()) {
09 // use iTree to filter by i
10 r' = filter(iTree, i);
11 // det. measures def. by OF
12 ts.set(i, calc(i, r', q.meas());
13 }
14 return ts;
15 }
```

As shown, the algorithm filters the records according to the **IN**- and **WHERE**-clause and creates an INTTREE for the filtered records (row 04). When iterating over the defined ranges, the created iTree is used to retrieve the relevant records for each range (row 08).

4.2 Performance

We ran several tests on an Intel Core i7-4810MQ with a CPU clock rate of 2.80 GHz, 32 GB of main memory, an SSD, and running 64-bit Windows 8.1 Pro. As Java implementation, we used a 64-bit JRE 1.6.45, with XMX 4,096 MB and XMS 512 MB. We tested the parser (implemented using ANTLR v4) and processing considering correctness. In addition, we measured the runtime performance of the processor for the three introduced algorithms (cf. Section 4.2), whereby the data and structures of all algorithms were held in memory to obtain CPU time comparability. We used a real-world data set containing 1,122,097 records collected over one year. The records have an average interval length of 48 minutes and three descriptive values: person (cardinality: 713), task-type (cardinality: 4), and work area (cardinality: 31). The used time-granule was minutes (i.e. time cardinality: 525,600). We tested the performance using the **SELECT TIMESERIES** queries shown in Table 2. Each query specifies a different type of query (i.e. different measure, usage of groups, or filters) and was fired 100 times against differently sized sub-sets of the real-world data set (i.e. 10, 100, 1,000, 10,000, 100,000, and 1,000,000 records).

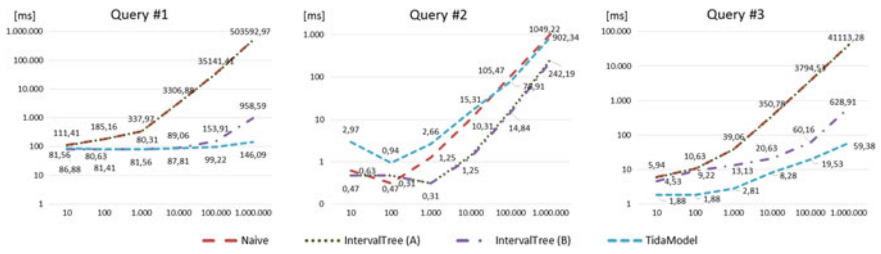


Fig. 14 The measured average CPU-time performance (out of 100 runs per query)

Table 3 Statistics of the test results

in DB	Number of records			Selectivity		
	Selected by query			Selected / in DB		
	#1	#2	#3	#1	#2	#3
10^1	1	0	0	0.1000	0.0000	0.0000
10^2	5	0	7	0.0500	0.0000	0.0700
10^3	12	2	46	0.0120	0.0020	0.0460
10^4	147	9	480	0.0147	0.0009	0.0480
10^5	1.489	121	5.148	0.0149	0.0012	0.0515
10^6	15.378	1.261	51.584	0.0154	0.0013	0.0516

The results of the runtime performance tests are shown in Fig. 14. As illustrated, the bitmap-based implementation performs better than the naive and INTTREE algorithms when processing query #1 and #3. Regarding query #2 the INTTREE-based implementations perform best. As stated in Table 3, the most important criterion to determine the performance is the selectivity. Regarding a low selectivity the INTTREE-based algorithm (B) performs best.

Nevertheless, considering persistency and reading of records from disc the algorithm might perform worse. We would also like to state briefly, that other factors (e.g. kind of aggregation operators used) influence the performance of the bitmap algorithm, so that it outperforms the INTTREE-based implementation, even if a low selectivity is given.

4.3 *Considerations*

The query language and processing introduced in this paper, is currently used within different projects by analysts and non-experts of different domains to analyze time-interval data. In the majority of cases, the introduced language and the processing is capable of satisfying the user's needs. Nevertheless, there are limitations, issues, and preferable enhancements. In the following, we introduce selected requests/improvements:

1. The presented query language and its processing do not support any type of transactions. A record inserted, updated, or deleted is processed by the system as an atomic operation. Nevertheless, roll-backs needed after several operations have to be performed manually. This generally increases implementation effort on the client-side.
2. The presented XML definition of dimensions (cf. Section 3.3.1) uses regular expressions to associate a member of a level to a descriptor value. Regular expressions are sometimes difficult to be formalized (especially for number ranges). An alternative, more user-friendly expression language is desired.
3. The **UPDATE** and **DELETE** commands (cf. Section 3.4.1) need the user to specify a record identifier. The identifier can be retrieved from the resultset of an **INSERT**-statement or using the **SELECT RECORDS** command. Nevertheless, users requested to update or delete records by specifying criteria based on the records' descriptive values.
4. When a model is modified, it has to be loaded to the system as new, the data of the old model has to be inserted and the old model has to be deleted. Users desire a language extension, allowing to update models. Nevertheless, the implications of such a model update could be enormous.

5 Conclusion

In this paper, we presented a query language useful to analyze time interval data in an on-line analytical manner. The language covers the requirements formalized by several business analyst from different domains, dealing with time interval data on a daily basis. We also introduced four different implementations useful to process the most frequently used type of query (i.e. **SELECT TIMESERIES**).

An important task for future studies is to confirm, or define new models and present novel implementations solving the problem of analyzing time interval data. In addition, future work should focus on distributed and incremental query processing (e.g. when rolling-up a level). The mentioned considerations (cf. Section 4.3) of our introduced language and its implementation should be investigated. Another interesting area considering time-interval data is on-line analytical mining (OLAM). Future work should study the possibilities of analyzing aggregated time series to discover knowledge about the underlying intervals. Finally, an enhancement of the processing of the two-step aggregation technique should be considered. Depending on the selected aggregations an optimized processing strategy might be reasonable.

References

1. E. Codd, S. Codd, C. Salley, *Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate*. 1993. E. F. Codd and Associates (sponsored by Arbor Software Corp.)
2. J. Mazón, J. Lichtenbörger, T. J., Solving summarizability problems in fact-dimension relationships for multidimensional models. In: *11th Int. Workshop on Data Warehousing and OLAP (DOLAP '08). Napa Valley, California, USA, 26.–30. October*. 2008, pp. 57–64
3. R. Kimball, M. Ross, *The data warehouse toolkit: The definitive guide to dimensional modeling*, 3rd edn. Wiley Computer Publishing, 2013
4. J. Allen, Maintaining knowledge about temporal intervals. Communication ACM **26** (11), 1983, pp. 832–843
5. P. Meisen, D. Keng, T. Meisen, M. Recchioni, S. Jeschke, Bitmap-based on-line analytical processing of time interval data. In: *12th Int. Conf. on Information Technology. Las Vegas, Nevada, USA, 13.–15. April*. 2015
6. P. Meisen, T. Meisen, M. Recchioni, D. Schilberg, S. Jeschke, Modeling and processing of time interval data for data-driven decision support. In: *IEEE Int. Conf. on Systems, Man, and Cybernetics, San Diego, California, USA, 04.–08. October*. 2014
7. M. Böhlen, B. R., J. C. S., Point-versus interval-based temporal data models. In: *14th Int. Conf. on Data Engineering, Orlando, Florida, USA, 23.-27. Feburary*. 1998, pp. 192–200
8. P. Papapetrou, G. Kollios, S. S., G. D., Mining frequent arrangements of temporal intervals, knowledge and information systems **21** (2), 2009, pp. 133–171
9. F. Mörchen, Temporal pattern mining in symbolic time point and time interval data. In: *IEEE Symp. on Computational Intelligence and Data Mining (CIDM 2009), Nashville, Tennessee, USA, 30. March–2. April*. 2009
10. F. Höppner, F. Klawonn, Finding informative rules in interval sequences. In: *IDA2001. LNCS*, vol. 2189, ed. by F. Hoffmann, N. Adams, D. Fisher, G. Guimarães, D. Hand, Springer, Heidelberg, 2001, pp. 123–132
11. A. Kotsifakos, P. Papapetrou, V. Athitsos, Ibsm: Interval-based sequence matching, 13th siam int. conf. on data mining (sdm13), austin, texas, usa, 02.–04. may. 2013
12. Y. Chen, M. Chiang, M. Ko, Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications **25** (3), 2003, pp. 343–354
13. R. Agrawal, R. Srikant, Mining sequential patterns. In: *Int. Conf. Data Engineering, Taipei, Taiwan*. 1995, pp. 3–14
14. P. Papapetrou, G. Kollios, S. S., D. Gunopulos, Discovering frequent arrangements of temporal intervals. In: *5th IEEE Int. Conf. on Data Mining (ICDM'05), IEEE Press*. 2005, pp. 354–361
15. F. Mörchen, A better tool than allen's relations for expressing temporal knowledge in interval data. In: *12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Philadelphia, Pennsylvania, USA*. 2006
16. C. Chui, B. Kao, E. Lo, D. Cheung, S-olap: An olap system for analyzing sequence data. In: *ACM SIGMOD International Conference on Man-agement of Data, Indianapolis, Indiana, USA*. 2010
17. M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, A. Mehta, E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In: *ACM SIGMOD International Conference on Management of Data, Athens, Greece*. 2011
18. B. Bebel, M. Morzy, T. Morzy, Z. Królikowski, R. Wrembel, Olap-like analysis of time point-based sequential data. In: *Advances in Conceptual Modeling*, ed. by S. Castano, P. Vassiliadis, L. Lakshmanan, M. Lee, 2012. 978-3-642-33998-1
19. C. Koncilia, T. Morzy, R. Wrembel, E. J., *Interval OLAP: Analyzing Interval Data, Data Warehousing and Knowledge Discovery (DaWaK 2014)*, vol. 8646. Springer Int., 2014
20. N. Kline, R. Snodgrass, Computing temporal aggregates. In: *11th Int. Conf. on Data Engineering (ICDE 1995), Taipei, China, 06.–10. March*. 1995, pp. 222–231
21. D. Rafiei, A. Mendelzon, Querying time series data based on similarity. IEEE Transactions on Knowledge and Data Engineering **12** (5), 2000

22. G. Spofford, S. Harinath, C. Webb, D.H. Huang, F. Civardi, *MDX-Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. John Wiley & Sons, 2006
23. T. Pedersen, Aspects of data modeling and query processing for complex multidimensional data. Ph.D. thesis, Aalborg Universitetsforlag, Aalborg, Department of Computer Science, Aalborg Univ., 2000. No. 4
24. H. Kriegel, M. Pötke, T. Seidl, Object-relational indexing for general interval relationships. In: *7th Int. Symposium on Spatial and Temporal Databases (SSTD 2001), Los Angeles, California, 12.–15. July*. 2001, pp. 522–542