Journal of Web Systems and Applications (2017) Vol. 1, Numuber 1
Clausius Scientific Press, Canada

ViMDQL: An Easy-to-Use Drag-and-Drop Visual Query Composer for Multidimensional Data

Sheng Liang, Bing Fang

[1]College of Mathematics and Information Science, Guiyang University, Guiyang, China

171380401@qq.com, 11676058@qq.com

Keywords: Query Composer, Multidimensional Data, Visual Query Builder

Abstract. In this study, we present ViMDQL, a useful system to tack the challenge of composing multidimensional data based analytical queries easily. Currently, fundamental query blocks such as import and export, create, retrieve, update, delete are supported, which can be used to loading and exporting data, creating, retrieving, updating, join, sampling and removing multidimensional data. Analytic functionalities such as aggregation, statistics are also supported. We demonstrated that, since ViMDQL make users can express their query intent by drag and drop to link query blocks together, which enable users can easily compose queries like the Stacker Game, it has been proved to be a productivity tool for graphically building multidimensional data based queries.

1. Introduction

Complex analysis over extremely large data set has become a challenge for both databases communities. In the big data era, people often face an ingestion of Terabytes of data on a daily basis, and often struggle to do in-depth analysis using the systems or tools deployed over their data. They eagerly need effective tools to enable them can easily express their complex analytical needs. The SQL technique of Relational database management systems is one of such tools which can represent complex analytical tasks. However, most complex data isn't naturally modeled as relational tables, meanwhile, most complex analytics, such as machine learning and predictive modeling, are best expressed as sequences of linear algebra operations over arrays like multidimensional data structures [1], e.g., GIS, astronomy, bioinformatics, LBS (Location based Services). Although multidimensional computational paradigm is prevalent in many domains, due to the lack of effective tools, it's still to be difficult for data analyst to write complex analytical task expressions.

The mismatch between application needs and database technology has a long history. Databases system often require the users must be skilled in SQL or other database programming languages. Furthermore, the syntax and semantics of these SQL-like language are difficult to learn for end-users. The situation in multidimensional databases however, is even more complicated, because they often contain complex data, and queries often tend to be excessively verbose and complicated. For the end-users who haven't computer programming expertise, when ask them formulate the analytical task into textural queries language statement, meanwhile, without any iconic or spatial clues to help them during composing, traditional textural query languages like SQL are tend to be not friendly enough, and the query composing can even become a hard task.

We believe that a standardized multidimensional data query language will be generalized like SQL and will also like "operator language", which defined a collection of primitives. For instance, multidimensional database system SciDB [1] implements both SQL-like Query Language (AQL) and Functional Language (AFL). In our work, we choose SciDB as the backend database engine of our ViMDQL system. We present a novel graphic building-block query builder, called "ViMDQL", to help users compose complex query for multidimensional data. We will illustrated ViMDQL in following sections of this article.

2. Related Work

In data processing area, form-based interface is exceedingly popular. For example, user can look up a pending airline reservations by inputs a frequent flyer number into a form. In scientific area, there is also some examples using form-based interface, e.g., In SkyServer [2], user can build a radial query by inputting a radius or search the objects by magnitudes. But this sort of form-based user interface rarely can express complex query. Data analyst usually want a visualization system which can facilitate them express their complex analytical task clearly.

Query-By-Example [3] is a good solution for non-sophisticated user, but it is limited by domain knowledge to set sample queries, and cannot support user-defined queries very well. There are already lots of works using visual interface with "Drag-and-Drop" graphic icons or objects to formulate general SQL queries for RDBMS, such as Ajax Query Builder [4]. But in multidimensional data store area, many efforts such as SPARQLGraph [5] can be seen as domain-specific methodology. NaLIR [6, 7] now can accept a logically complex English language sentence as query input for data store, but there is still a long way to take natural language queries as a database query interface due to the difficulty of multilingual user-specified query translation into the actual schema structure in the multidimensional data store.

ViMDQL is our effort to offers graphic query builder capability for multidimensional data store. Nearly all the common features needed for visually compose an analytical query are included. In ViMDQL, if two blocks cannot match, you can never put them together. The blocks construction will be translated into native query language (such as SQL for RDBMS and AQL/AFL for SciDB) for underlying query execution.

3. System Overview

An overview of ViMDQL system architecture is given in Fig. 1. The entire system has 5 components. The Query Builder offers a graphic workspace for users to edit their queries. The Query Translator is responsible for taking linked blocks as input and generating the correct textual query statements. The Workflow Manager extends four patterns workflow control blocks to Builder. Shared queries are saved in the Query Library. The Database Connector used to connect the multidimensional data storage engine, and then issue analytical queries and get the results back.

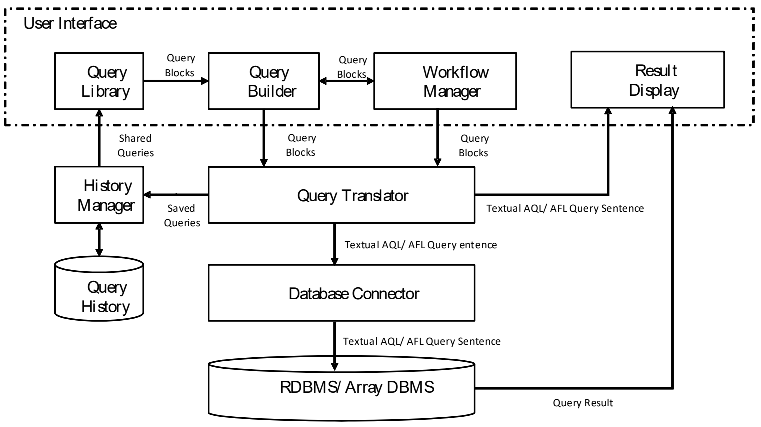


Fig.1. Architecture of ViMDQL

Query Builder. It provide a workspace to graphically compse analytical queries. We customize a basic set of blocks for manipulating multidimensional data. These blocks are divided into five categories: create, retrieve, update, delete, import and export, including creating and removing arrays, loading data, transferring data, updating arrays, and other basic array tasks such as selecting, joins, sampling. Analytic functionalities such as aggregation, statistics, and nested query are also supported.

ViMDQL runs in a web browser. It uses blocks that link together to make writing query statements easier. Users just need to drag and drop blocks to link together and tune or set parameters to express their query intent. We set syntax check through type check mechanism. Each block is labeled with strict input-type and out-type information so that invalid statements combination will be failed. If user are trying to put two mismatch blocks together, Builder will give instantaneous feedback.

Query Translator. It collect all the arguments and field data form user-defined blocks in the Builder workspace, and then utilizes blocks' organization structure to generate appropriate target statements of query language expressions. In current ViMDQL, it support translate query blocks into SciDB AQL and AFL statements. The output of Query Translator, textual query statements can be immediately displayed or be issued to underlying DBMS through Database Connector.

Query Library. Users can save their query statement or share it with others. The Query Library saves shared queries, organizes content by type and category and provides search capabilities to find the appropriate statement. Specially, users can obtain existing query from Query Library and drag it into workspace to edit into their own queries.

Workflow Manager. ViMDQL use a procedural composition mode. Users build their query step by step just like a sequence workflow execution mode. In order to make users' data exploration easier, ViMDQL add some common workflow control blocks in Workflow Manager. Currently, Workflow Manager covers four patterns [8]: sequence, parallel split, synchronization and exclusive choice. Their usages are as same as the query blocks. Users can directly drag them into the workspace to organize the prepared queries.

Database Connector. Once users choose to run the successfully composed query over certain DBMS, we need to get connection with database firstly. That's what Database Connector does. It is used to establish database connection, get query statement form Query Translator, issue query and return the results. In the currently implementation, we take SciDB as the database engine, and other database will be supported soon.

4. User Interface of ViMDQL

ViMDQL's user interface is depicted as Fig. 2. It mainly consists of a toolbar, which holds all the available blocks and a workspace, where users place the query blocks. Since current ViMDQL use SciDB as the backend multidimensional data storage engine, it support both AQL and AFL, and users can page by clicking the corresponding labeled button, users will attain the matching language sentences which are translated from their building-blocks. At the history page or result page, the history or result will be displayed.

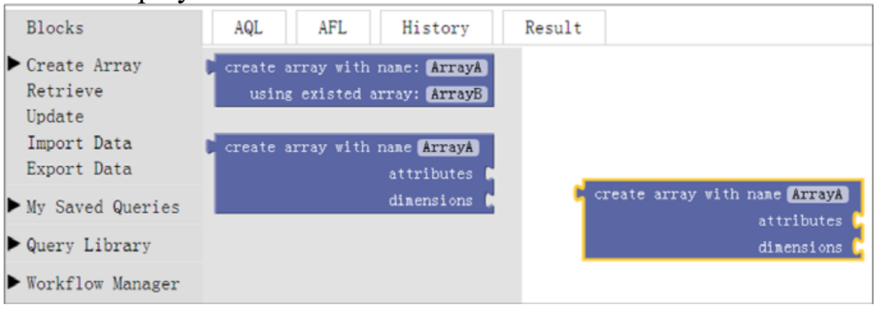


Fig.2. User Interface of ViMDQL

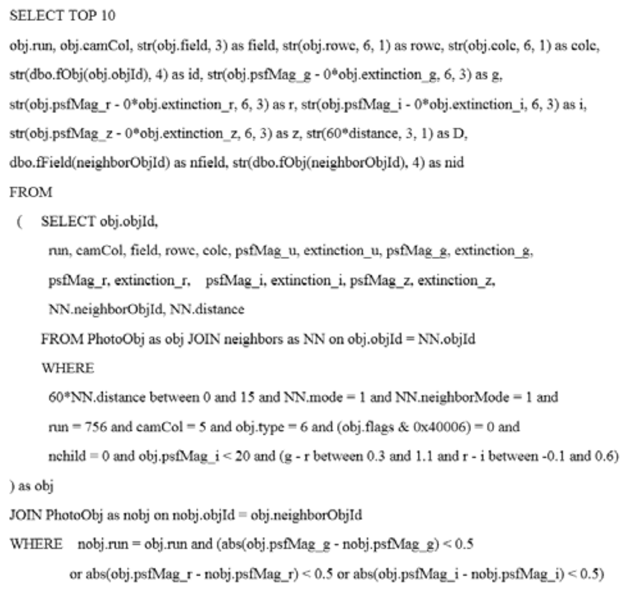
In traditional scenarios, it's a very difficult task for inexperienced users successfully write a complex nested query statement without syntax mistakes. In ViMDQL, users can get feedback immediately for his work, and it make users can easily compose nested complex queries with the intuitional user interface and its underlying interactive design.

Since it often to be waste of time to write repetitive statements. Users can save and share their work to others. One can search similar shared queries by keyword or by domain in the Query Library, and then drag the shared queries into the workface to customize it into their own query task.

5. Usage Demonstration

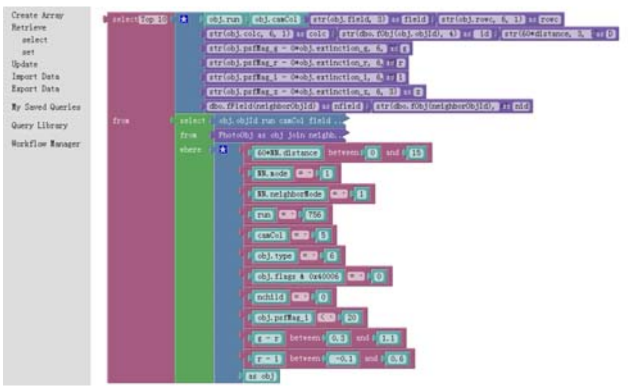
In this section, we demonstrate ViMDQL's capability by employ it in massive astronomic data analytic task. In this scenario, we use Sloan Digital Sky Server Data Release 9 as our test dataset and choose a set of queries from SkyServer [2] as they're typical and representative sample queries in astronomy. Fig. 3 shows a nested AQL query statement that merges all galaxies and outputs the total number of galaxies. Actually, we can conclude it as "SELECT * FROM * SELECT * FROM * JOIN * ON * WHERE * JOIN * WHERE" construction mode. Imagine that users are inexperienced and need to build such a complex query manually, it won't be an easy task. In this case, users have lots of parameters to set up and should be more careful to avoid syntax mistakes.

Figure 4 illustrated how to build the aforementioned complex nested query with ViMDQL. Users just need to drag and drop appropriate blocks into the workspace and link them together. They don't have to be worried about syntax correctness because system will immediately give feedback if they make a wrong block. Since there is no possibility to link two mismatch blocks together in ViMDQL, it is relative easier for imperienced users to compose such a complex query.



```
SELECT TOP 10
obj.run, obj.camCol, str(obj.field, 3) as field, str(obj.rowc, 6, 1) as rowc, str(obj.colc, 6, 1) as colc,
str(dbo.fObj(obj.objId), 4) as id, str(obj.psfMag_g - 0*obj.extinction_g, 6, 3) as g,
str(obj.psfMag_r - 0*obj.extinction_r, 6, 3) as r, str(obj.psfMag_i - 0*obj.extinction_i, 6, 3) as i,
str(obj.psfMag_z - 0*obj.extinction_z, 6, 3) as z, str(60*distance, 3, 1) as D,
dbo.fField(neighborObjId) as nfield, str(dbo.fObj(neighborObjId), 4) as nid
FROM
 (   SELECT obj.objId,
       run, camCol, field, rowc, colc, psfMag_u, extinction_u, psfMag_g, extinction_g,
       psfMag_r, extinction_r,   psfMag_i, extinction_i, psfMag_z, extinction_z,
       NN.neighborObjId, NN.distance
     FROM PhotoObj as obj JOIN neighbors as NN on obj.objId = NN.objId
     WHERE
       60*NN.distance between 0 and 15 and NN.mode = 1 and NN.neighborMode = 1 and
       run = 756 and camCol = 5 and obj.type = 6 and (obj.flags & 0x40006) = 0 and
       nchild = 0 and obj.psfMag_i < 20 and (g - r between 0.3 and 1.1 and r - i between -0.1 and 0.6)
) as obj
JOIN PhotoObj as nobj on nobj.objId = obj.neighborObjId
WHERE    nobj.run = obj.run and (abs(obj.psfMag_g - nobj.psfMag_g) < 0.5
           or abs(obj.psfMag_r - nobj.psfMag_r) < 0.5 or abs(obj.psfMag_i - nobj.psfMag_i) < 0.5)
```

Fig. 3. AQL statements of sample query



```
select Top 10
  obj.run  obj.camCol  str(obj.field, 3) as field  str(obj.rowc, 6, 1) as rowc
  str(obj.colc, 6, 1) as colc  str(dbo.fObj(obj.objId), 4) as id  str(60*distance, 3, as D
  str(obj.psfMag_g - 0*obj.extinction_g, 6, as g
  str(obj.psfMag_r - 0*obj.extinction_r, 6, as r
  str(obj.psfMag_i - 0*obj.extinction_i, 6, as i
  str(obj.psfMag_z - 0*obj.extinction_z, 6, 3) as z
  dbo.fField(neighborObjId) as nfield  str(dbo.fObj(neighborObjId), as nid
from
  select  obj.objId run camCol field ...
  from    PhotoObj as obj join neighb...
  where
    60*NN.distance between 0 and 15
    NN.mode = 1
    NN.neighborMode = 1
    run = 756
    camCol = 5
    obj.type = 6
    obj.flags & 0x40006 = 0
    nchild = 0
    obj.psfMag_i < 20
    g - r between 0.3 and 1.1
    r - i between -0.1 and 0.6
  as obj
```

Fig. 4. Sample Query Blocks of ViMDQL

Acknowledgements

This work was supported by Science and Technology Fund of Guizhou Province (No. J-LKJ[2013]49), Science and Technology Fund of Department of Education of Guizhou Province (No. KY[2013]110).

References

[1] M.Stonebraker, P.Brown and et al.: *SciDB: A database management system for applications with complex analytics*. Computing in Science & Engineering, Vol.15(3), p.54-62 (2013)
[2] The Sloan Digital Sky Survey SkyServer, http://skyserver.sdss.org
[3] M.Yen and R.Scamell: *A human factors experimental comparison of SQL and QBE*. IEEE Transactions on Software Engineering, Vol.19(4), p.390-409 (1993)
[4] Ajax Query Builder: http://www.ajaxquerybuilder.com/
[5] D.Schweiger, Z.Trajanoski and et al.: *SPARQLGraph: a web-based platform for graphically querying biological Semantic Web databases*. BMC bioinformatics, Vol.15(1), p.279 (2014)
[6] F.Li, and H.Jagadish: *NaLIR: an interactive natural language interface for querying relational databases*. In Proceedings of the 2014 ACM SIGMOD, p.709-712 (2014)
[7] F.Sébastien: *Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language*. Semantic Web, Vol.8(3), p.405-418 (2017)
[8] A. Der, W. Hofstede, A.Kiepuszewski, and A.Barros: *Workflow patterns*. Distributed and parallel databases, Vol.14(1), p.5-51 (2013)