XOQL: OBJECT QUERY MARKUP LANGUAGE

Pavel P. OLEYNIK
System Architect Software, Aston Company; Associate Professor, Shakhty Institute (branch), Platov South Russian State Polytechnic University (NPI)
Address: 1, Lenin square, Shakhty, Rostov Region, 346500, Russian Federation
E-mail: xsl@list.ru

Modern corporate information systems (CIS) are designed by employing object-oriented paradigm and concepts. This approach is often applied both to implement client applications and to build a server component (target DBMS). The application of object-oriented design pattern in software development enables to save business objects from RAM to persistent memory. This paper focuses on XOQL (XML Object Query Language) - an object query language that uses XML to describe syntax. This article presents a deep and comprehensive review of existing publications. Abundance of examples enables to demonstrate various currently available languages.

This paper suggests a feasible option to present basic syntactic constructions of object query language in the form of XML-documents. Prior to syntax design the optimality criteria have been formulated (these are described in detail in this paper). Query language syntax extensions are outlined in addition to basic ones, as well as extension approaches by involving proprietary constructions. An optimal language structure is presented accompanied by descriptions of tags, attributes and admissible values. At the end of this article there are plenty of examples of various common queries.

Key words: object query language, corporate information systems, query language, markup language, XML, databases.

Citation: Oleynik P.P. (2015) XOQL: Object Query Markup Language. *Business Informatics*, no. 2 (32), pp. 30–38.

1. Introduction

Extensible Markup Language (XML) is used to represent semi-structured data, i.e. information, which structure is unknown or is anticipated to face profound future changes [1]. Due to its flexibility, visibility and existence of a large number of supporting technologies this language is utilized in different parts of a corporate information system (in server database, client application, server applications). One of tasks requiring retention of information, which structure is unknown in advance, is a task to present syntax of a data query language (QL) (for example, SQL, OQL, LINQ, etc.). Implementation of these QL as XML-documents will enable to unify the data manipulation process, which in this case does not depend on any specified data source (file system, RDBMS, OODBMS, etc.).

The suggested syntax may be used at the transport level to transmit queries between levels and tiers in a complex information system. To generate queries and to perform syntax parsing one may use third-party libraries, which are available on target client or server platform.

Classic implementation of a syntax representation of a query language in a software application is in the form of text strings containing key words [2], pertaining to respective grammar. In doing so, full implementation of such QL entails (according to the author) the following disadvantages:

1. The necessity to carry out a comprehensive semantic analysis for syntax by itself. Interdemand to develop strings handling classes, which will check a particular query for grammar and syntax to build a tree for added tokens of a query language.

2. The necessity to build a class hierarchy to generate a query construction. Modern applications are designed on object-oriented programming languages (OOPL) and manipulate instances of classes, calling their methods and assigning attribute values. It is best to use the generation of a query by creating a class instance (object) of syntax constructions.

Using XML to present a query language syntax would allow to avoid the above listed disadvantages and independent language from specific DBMS used to store information.

The plurality of XML analyzers and parsers for client and server applications enables to simplify the process of syntactic and semantic analysis. These parsers have an object model to generate programmatically the structure of XML documents, which constitute an object query in this case.

2. Related papers and technologies

All existing literature can be divided into three major categories:
1. Papers, which describe implementation of query languages in object-relational mapping (ORM) tools;
2. Articles, which deal with domain-specific query language, research in certain data domains;
3. Publications, which specify ways of using XML-documents in certain syntax constructions of query languages.

Each of these categories is described in detail in the following subsections.

2.1. Query languages, implemented in ORM-tools

Currently object-relational mapping (ORM) tools have been increasingly used in information system design to implement an object system in relational DBMS. Every ORM-tool implements its own object query language dialect. For example, Hibernate software product supports HQL and incorporates a rich class object model designed to generate a query [14].

HQL-query is recorded in the <query> tag in the CDATA section, and its structure is similar to SQL. It does not use the possibility of organizing a hierarchy of nodes for a clearer presentation of a structure and syntax of an object query, i.e. HQL developers have resorted to classical implementation, as described above, involving a syntax analysis and verification of certain strings of defined grammar; a built-in hierarchy of classes has been developed as well to support a query generation.

Enterprise JavaBeans (EJB), a technology that can be regarded as ORM-tool implemented on Java language, supports EJB-QL query language that enables various manipulations with objects (to select, to insert, to update, to delete, etc.) [12]. It is possible to write EJB-QL-query in a configuration file (in the descriptor deployment) that is a XML-document.

A query that uses only one <ejb-ql> tag consists of strings of SQL-like QL. Therefore, it suffers from the above listed drawbacks.

2.2. Domain-specific query languages

Query language design issues are discussed in theoretical papers dealing with object-relational mapping patterns. Article [13] describes QL implementation that enables to select class instances from a RDBMD-based object system, and a query can be built based on a XML-schema published in the above mentioned paper.

This implementation makes it possible to extract main temporal data, and it is used in a specific data domain (medical IS). Therefore, the language is not orthogonal to the data domain and cannot be used as the main information system in a different profile.

It is possible to include a specific language — CAML (Collaborative Application Markup Language) into this category of papers; it is presented in [21-22] and employs Windows SharePoint Services that supports a single cross-enterprise electronic workflow.

To extract data the <Query> tag is used; the <Where> inner node enables to create complex expressions with a combination of comparison and logical operations. In each comparison operation the <FieldRef> tag is utilized to indicate links in a field, and the <Value> node presents a comparable value. Note that the <Value> node can contain either a constant or a field name. In this complex arithmetic expression a similar text string should be specified. More correctly, this needs to be done with nested tags, thus providing an opportunity to describe complex expressions.

Let's consider another domain-specific markup language. KDDML (KDD Markup Language) is used as an intermediate language to extract information through knowledge discovery CISs. [23-24].

To formulate a query the <KDD_QUERY> root tag is used; it enables to identify a query unambiguously by using a name specified in the «name» attribute. Then, this query can be executed using a name in the <CALL_QUERY> tag. To determine a structure of decision trees generated for data analysis and extraction the <TREE_MINER> XML-element is utilized. The xml_dest attribute specifies a location, where output data will be saved. Therefore, this language can only be used in knowledge-based decision support systems. Substantive findings of the literature review suggest that a domain-specific query language cannot be used as a uniform data query language.

2.3. Methods for presentation of certain syntactic constructions of query languages as XML

Papers within this category most closely correspond to tasks addressed in this article, because these offer a unified approach to certain syntactic constructions of QL. This problem has been solved in XSQL [20], where a markup language has been designed to describe SQL-query structures.

The root element is the <xsql> tag, containing one or more queries that are described in the <query> tag. First of all, we are interested in a set of tags, which are used to extract data, therefore, let's consider the <select> child nodes tag. The <table_column> node indicates a field name (<column>) and a corresponding table name (<table>). XSQL doesn't offer any simple way to specify columns to be extracted and to set alias for a field. To specify conditions for data filtering the <conditional> tag is used that contains predicates in the form of string expressions.

To create complex queries and queries with parameters the <statement> tag that contains the SQL-query is utilized. This approach to query description suffers from the above mentioned disadvantages relating to ORM-tools (Sec. 2.1).

Unfortunately, at present the XSQL project is no longer being developed and now only dtd-description of some constructions is available, that is clearly not enough to describe queries for a real-world application.

Another approach developed for submission to logical conditions for SQL language filter (for submission of design directives) is described in [25] that deals with SIML (Software Integration Markup Language).

SIML uses the <sql> tag that specifies a list of fields and tables, which information must be selected to be written as a SQL-string. This approach has the above listed disadvantages.

Use the <filters> child XML-node that contains conditions for data filtering. Each individual comparison condition is created by the <query> tag: a field name is specified in the <field> tag, and a value is indicated in the <value> tag.

Basically, [25] describes equality comparison operations only. Construction of predicates, containing complex logical expressions with multiple operations, is not covered in the above mentioned paper. Also, an idea of two opposite approaches to description of selected fields (as a string SQL-query) and filtering conditions (as a set of nested tags) constitutes a serious disadvantage, that manifests itself in the impossibility of writing a query for data extraction from a number of tables and implementation of join – operations. These deficiencies have resulted from the fact that SIML had been designed as a language aimed to integrate data stored in different sources, and not as a QL.

A study, where SQL syntax presentation principles have been elaborated most comprehensively, refers to ZsqlML project (Zenark's XML for SQL) [26].

Though ZsqlML offers a number of advantages (like all other languages, outlined earlier in this section, involving nested SQL constructs), it has a huge drawback: it is intended for submissions to SQL syntax that lacks many object extensions and is used often to develop object-oriented applications.

Summarizing the review of all three categories of papers, one could argue that each of the considered approaches (and tools) has a number of disadvantages. The following sections of this paper describe an implemented query language syntax that lacks the above mentioned weaknesses.

3. Implementation of XOQL query language

Modern corporate information systems (CIS) are designed by employing the object-oriented paradigm. To develop an objective data domain model software application that enables to save business object to persistent memory the following approaches are used often:

1. The use of an object-oriented DBMS for data storage. The functional capabilities of object-oriented DBMSs are specified in ODMG 3.0 Standard [16-17].
2. Implementation of application's business logic in object-relational DBMS. Object extensions of relational DBMSs are regulated by SQL:2003 Standard [10].
3. Implementation of an object system in relational DBMS environment (object relational mapping, ORM) [13, 15].

Every tool that implements one of these approaches provides the developer with a specified query language. In author's opinion, the most developed one is the object query language (OQL, Object Query Language). Its latest specification is presented in «The Object Data Standard: ODMG 3.0» [16]. At the same time, each manufacturer of an object-oriented DBMS (OODBMS) supports a subset of language constructions (their own dialect) [17].

3.1. Optimality criteria to be implemented in a query language

The main advantage of XML is the availability of a large number of interrelated technologies that define and control the syntax (semantic) structure of an application (language), which creation doesn't necessitate any partial parsing, as this case implies a performed parser [3]. Modern OO-languages include a class library and XML parser [3-6]. In addition, the latest versions of popular DBMS, such as Oracle, MS SQL Server and DB2, support embedded data XML type that is currently included in SQL 2003 standard [7-10].

To define valid structures of QL to be followed by a syntax analysis (in our case – parsing) various technologies can be used, for example, by creating DTD-definitions or by involving XSD-(XDR-) schemes. If this is not enough, a syntactic (and semantic) analysis can be performed by using queries in XQuery language that enables to return individual elements (nodes) of a XML-document [11].

To build a semantic structure in XOQL optimality criteria need to be formulated, and these should correspond to QL implementation (OCQL):

1. Independence of a data domain pertaining to a developed software application. This will enable to unify QL and to apply it in any information system. Therefore, it is necessary to identify the structure of common query language XML-nodes with such names as <Select>, <From>, <Where>, etc, as its name indicates that selected tags are not relevant to data domain, but describes various elements of QL.

2. A clear structure of a query, implemented with a set of nested XML-nodes. Nested XML-nodes are often used to ensure the most accurate display, in particularly, in the hierarchy of QL commands. For example, the <Where> element should be inside the <Select> element, as it indicates a data filtering condition and determines the Select type of a request (data extraction). Correctness of embedded XML-nodes can be checked during query parsing by employing dependent technologies, as described above.

3. The possibility to extend query language syntax by introducing new constructs using nodes and attributes. Because a query structure does not depend on a data source (OCQL4), the addition of any new tags (and attributes) does not affect applications that use older versions of the syntax, i.e. expansion will occur in accordance with the backward compatibility principles. Since a query is executed on a data source (e.g., RDBMS), procedures need to be developed to transform a XML-specific query into QL that is supported by a source. When adding a new element representing specific query language syntax a target DBMS requires adjustment procedures for transformation.

4. Availability of basic syntactic constructions, which are presented in Object Query Language (OQL). Key features of the object query language include: 1) a path expression to describe complex relationships between classes, and 2) possibility to extract as a part of a projection not only an atomic value from an attribute class, but whole objects (class instance). Its implementation (and a backlog) will provide a developer with a functional IS offering ample opportunities to select, update and delete data, similar to modern QL implemented in many popular OODBMS, ORM-tools and in distributed object applications.

5. Independence from any data source, data model and application architecture. Despite the fact that a query language should support syntactic constructions of an object query language (OQL), the same query can be used to manipulate (to select, to update, to delete) information physically stored in different data sources (OODBMS, ORM-tools, ORDBMS, etc.). For each data source the transformation procedure developed in the dialect supports QL. Given the magnitude of today's ERP, and the fact that client applications can simultaneously work with multiple DBMS, implementation of the above listed criteria will enable to reduce the cost of developed applications operating in heterogeneous environments. The language syntax does not depend on architecture of applications, and transformation in QL supported data source can be performed at any tier (layer) of a software application.

3.2. XOQL language syntax constructions

In order to determine allowable syntax semantics QL describes XML-schema (*Fig. 1*). The<XOQL> tag is rooted in any XOQL-query; it has an attribute version, designed to describe a version of a query language. The value of this attribute is analyzing of a program that carries out transformation of XOQL-query into QL supported by a target DBMS. This technique offers organized support for backward compatibility, which allows applications to deal with different versions of the language. The <Select> child element indicates that a query selects information from the repository. Since other types of queries (inserting, deletion and modification) are not considered in this paper, the relevant syntactic structure (presented by XML-tags) is absent in *Fig. 1.*

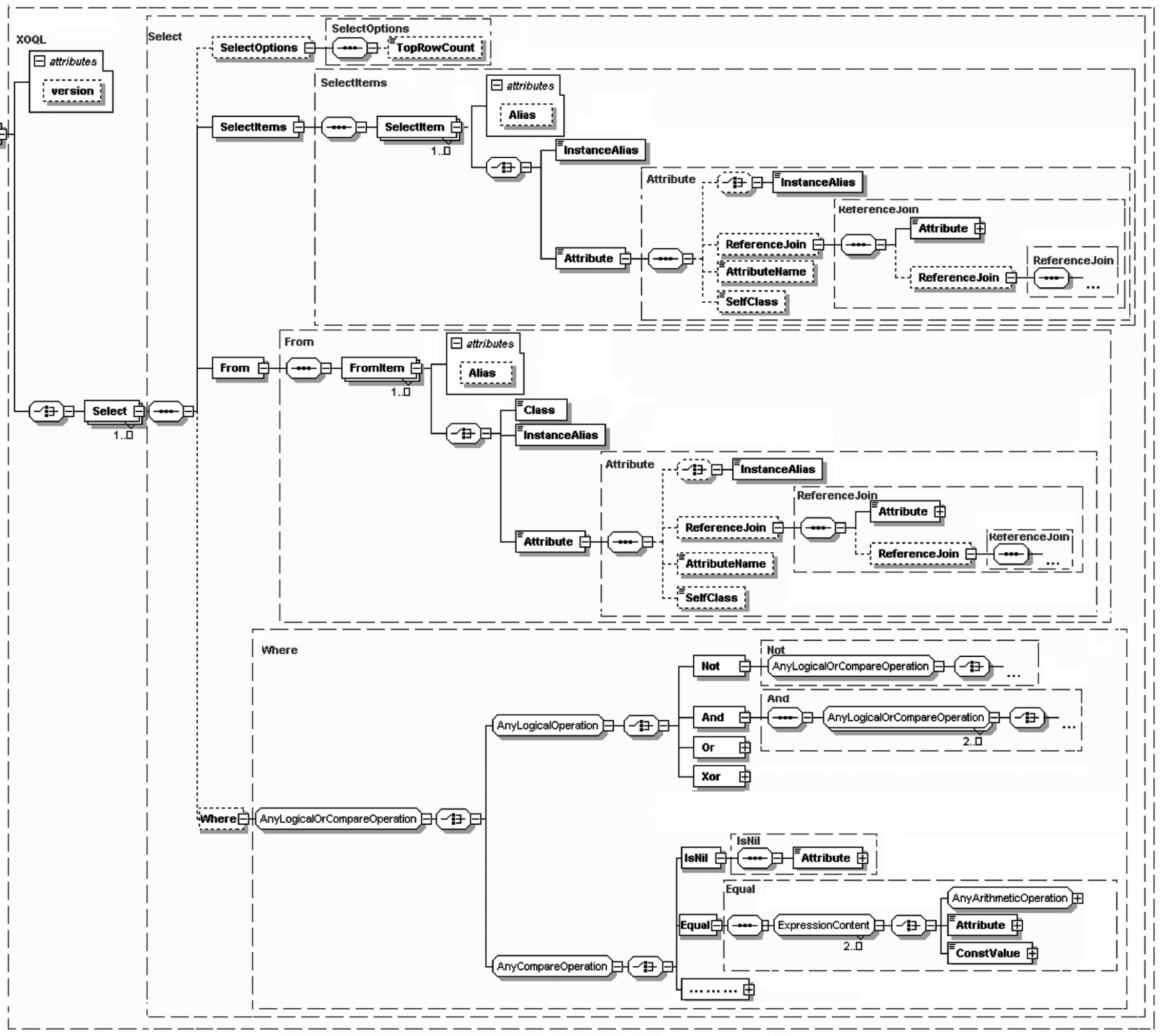

Fig. 1. XML-schema that describes basic syntactic constructions of XOQL

Inside the <Select> element the <SelectOptions> optional tag may be used to indicate various options affecting the data selection. For example, using the <TopRowCount> child node one can restrict the output set of the first N objects, where N is a positive integer. Composition and structure of available options depend on a query language supported by a target database, used as an information repository for application development.

To describe elements of a projection of a returned data set the <SelectItems> tag is used. Each element is defined in the <SelectItem> XML-node and may contain an alias (specified as the Alias attribute value) under which it will be returned to a client application, i.e. the Alias attribute is utilized as an equivalent to directives «as» in data selection (using the «select» operator) in SQL.

To extract objects as items of a projection the <InstanceAlias> tag is applied, it contains a collection of declared alias accessible through the <FromItem> node. To select data the <Attribute> tag is used. If an attribute is represented by an atomic literal data type (string, integer, the number of fixed-point, etc.) and is uniquely determined by its affiliation to any type of element collections, it is sufficient to indicate only its name (as the value of a XML-element). If the name of an attribute disallows to unambiguously determine an element type in a collection, that happens, when there are several attributes with identical names in one scope (of a variable), the <InstanceAlias> tag is employed to specify a desired alias collection. In this case the «attribute» name of a class is defined as the value of the <AttributeName> XML-element.

To extract an attribute value to describe an aggregated class of a written path expression the alternative nested tags (<Attribute> and <ReferenceJoin>) are used.

If a value to be selected must be assigned to any specific class (type) (in order to select an instance of a derived class by referring to a basic class), its name need to be inserted as the value of the <SelfClass> node.

The <From> tag allows to specify various collections (using <FromItem> nested nodes) in the Cartesian product that constitutes a source for data extraction. A collection can serve as an extent containing objects of a specified class and class description by using a path expression from a track before an assigned collection. In the first case, the <Class> tag, containing the name of a class is used to return objects, and in the second case, the <InstanceAlias> XML-node, containing the name of a alias assigned earlier for a particular collection, or the <Attribute> item, extracting an attribute type (class), is used. The «Alias» attribute is utilized to specify alias collections for subsequent references to value setting in the <InstanceAlias> tag.

To describe a logical predicate that imposes conditions for data extraction filtration the <Where> tag is used. The figure shows that a logical operation (and, or, not, xor), and comparison operations (> =, <, etc.) can be specified in this tag.

In view of what is stated in the foregoing it appears that XOQL does meet all optimality criteria formulated for the query language in sect. 3.1.

3.3. Test example: XOQL-query

Let's consider some XOQL-queries, which clearly demonstrate the underlying principles of object queries representation through XML. It is assumed that data is extracted from a test object model.

```
select c.Name as CompanyName,
c.Phone
       ,c.LegalAddress
from Company c, c.LegalAddress a
where a.City='Moscow'
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XOQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="XOQL.xsd">
  <Select>
    <SelectItems>
      <SelectItem Alias="CompanyName">
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>Name</AttributeName>
        </Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>Phone</AttributeName>
        </Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>LegalAddress</AttributeName>
        </Attribute>
      </SelectItem>
    </SelectItems>
    <From>
      <FromItem Alias="c">
        <Class>Company</Class>
      </FromItem>
      <FromItem Alias="a">
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>LegalAddress</AttributeName>
        </Attribute>
      </FromItem>
    </From>
    <Where>
      <Equal>
        <Attribute>
          <InstanceAlias>a</InstanceAlias>
          <AttributeName>City</AttributeName>
        </Attribute>
        <ConstValue>Moscow</ConstValue>
      </Equal>
    </Where>
  </Select>
</XOQL>
```

Fig. 2. Object query that has extracted information about companies registered in Moscow: QQL – at the left, XOQL – at the right

Let's consider a complex example (*Fig. 2*) of an XOQL-query involving extraction of name, contact phone number and registered office address of a Moscow company.

The «from» operator (the <FromItem> tag) indicates that data selection has been performed from two collections (extents), containing instances of the Company and Address classes, respectively. Aliases are assigned to extents («c» and «a», respectively), for which the «Alias» attribute has been used. Then the new name is used in the <InstanceAlias> tag with the declaration of the projection item of the resulting data set (the <Select> tag), and with reference to the previously announced collection of the <FromItem> XML-node. *Fig. 2* (at the right) demonstrates the «CompanyName» indication alias a for projection item under which it will be presented in the resulting dataset that is realized by assigning the «Alias» value attribute (the <SelectItem> tag).

A predicate that imposes restrictions on the resulting projection (to choose only organizations registered in Moscow) is specified by using the <Where> tag that contains a combination of logical, arithmetic and comparison operations. For comparison purposes, the «City» value attribute of the «CompanyAddress» class on equality constantly applies the <Equal> tag, where there are two nested nodes (<Attribute>, <ConstValue>) used to submit the name attribute and the «Moscow» constant, respectively.

These examples of XOQL-queries demonstrate the wide possibilities of developed QL and the use of syntactic structures, presented in the form of XML-tags and attributes.

Note that the resulting XOQL-query is more cumbersome than the original one and contains more than 40 lines of XML markup. However, the use of third-party libraries and parsers eases generation of such queries.

4. Conclusion and future research

Further research should concentrate on expansion of the proposed additional syntax constructions used for data selection (order by, group by, having, etc.) and description of directives to add new data (the insert operator), edit (the update operator) and remove (the delete operator) to existing data. In addition, syntactic constructions are needed to describe nested subqueries. Also, it is imperative to develop and to implement an algorithm to transform XOQL-queries into a certain dialect of a query language (e.g., SQL), supported by a specific DBMS. ■

References

1. Graves M. (2001) *Designing XML databases.* Prentice Hall PTR.
2. Aho A., Lam M., Sethi R., Ullman J. (2006) *Compilers: Principles, techniques and tools.* Addison Wesley.
3. Holzner S. (2003) *Real World XML (2nd Edition).* Peachpit Press.
4. Albahari J., Albahari B. (2012) *C# 5.0 in a Nutshell, 5th Edition. The Definitive Reference.* O'Reilly Media.
5. Perrone P.J., Venkata S.R., Chaganti K.R. (2000) *Building Java enterprise systems with J2EE.* Sams Publishing.
6. Wood K. (2003) *Delphi developer's guide to XML,* 2nd Edition. Wordware Publishing.
7. Wang J. (2011) *Oracle Database 11g Building Oracle XML DB Applications.* Oracle Press.
8. Mistry R., Misner S. (2014) *Introducing Microsoft SQL Server 2014.* Microsoft Press.
9. Mullins C. (2012) *DB2 Developer's guide – A solutions-oriented approach to learning the foundation and capabilities of DB2 for z/OS*, 6th Edition. IBM Press.
10. *SQL: 2003 specification.* Available at: http://www.wiscorp.com/sql_2003_standard.zip (accessed 10 January 2015).
11. *XQuery 3.0 use cases.* W3C Working Group Note 08 April 2014. Available at: http://www.w3.org/TR/xquery-30-use-cases/ (accessed 10 January 2015).
12. Rubinger A.L., Burke B. (2010) *Enterprise JavaBeans 3.1,* 6th Edition. O'Reilly Media.
13. Nadkarni P.M., Brandt C.A., Morse R., Matthews K., Sun K., Deshpande A.M., Gadagkar R., Cohen D.B., Miller P.L. (2003) Temporal query of attribute-value patient data: utilizing the constraints of clinical studies. *International Journal of Medical Informatics*, no. 70, pp. 59–77.
14. Leonard A. (2013) *Pro Hibernate and MongoDB.* Apress.
15. Fowler M. (2002) *Patterns of enterprise application architecture* (Addison-Wesley Signature Series). Addison-Wesley Professional.
16. Cattell R.G., Barry D.K. (2000) *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann Publishers.
17. Jordan D. (1998) *C++ Object Databases: Programming with the ODMG standard.* Addison-Wesley.
18. *Mathematical Markup Language (MathML) Version 2.0.* Second Edition. Available at: http://www.w3.org/TR/MathML2/chapter4.html#id.4.4.3 (accessed 10 January 2015).

19. Grand M. (1998) *Patterns in Java. Volume 1. A catalog of reusable design patterns illustrated with UML*. John Wiley & Sons.
20. *XSQL — Combining XML and SQL*. Available at: http://xsql.sourceforge.net/manual.php (accessed 10 January 2015).
21. *Collaborative Application Markup Language (CAML) structure specification*. Available at: http://download.microsoft.com/download/8/5/8/858F2155-D48D-4C68-9205-29460FD7698F/%5BMS-WSSCAML%5D.PDF (accessed 10 January 2015).
22. Fox S., Johnson C., Follette D. (2013) *Beginning SharePoint 2013 Development*. Wrox.
23. *KDDML: A middleware language and system for knowledge discovery in databases*. Available at: http://kdd.di.unipi.it/kddml/papers/kddml.pdf (accessed 10 January 2015).
24. *KDDML language: Reference guide*. Available at: http://kdd.di.unipi.it/kddml/downloads/documentazione/Specifiche/kddml_specification_2_0_16.pdf (accessed 10 January 2015).
25. Xu Y., Shi M. (2004) SQL Markup language for enterprise integration. Proceedings of the *2004 IEEE International Conference on Services Computing (SCC 2004), 15-18 September 2004, Shanghai, China,* pp. 413—416.
26. *ZsqlML (Zenark's XML for SQL)*. Available at: http://sourceforge.net/projects/zsqlml/ (accessed 10 January 2015).
27. Butek R. (2005) *Web services tip: Use polymorphism as an alternative to xsd:choice*. Available at: http://www.ibm.com/developerworks/webservices/library/ws-tip-xsdchoice.html (accessed 10 January 2015).

XOQL: ОБЪЕКТНЫЙ ЯЗЫК ЗАПРОСОВ

П.П. ОЛЕЙНИК

кандидат технических наук, системный архитектор программного обеспечения, ОАО «Астон»; доцент, Шахтинский институт (филиал), Южно-Российский государственный политехнический университет (НПИ) им. М.И. Платова

Адрес: 346500, Ростовская область, г. Шахты, пл. Ленина, д. 1

E-mail: xsl@list.ru

Современные корпоративные информационные системы (КИС) разрабатываются с применением объектно-ориентированной парадигмы и проектируются в понятиях объектно-ориентированного дизайна. Этот подход часто применяют как при реализации клиентского приложения, так и при создании серверного уровня, реализованного в среде целевой СУБД. Применение принципа предметно-ориентированного проектирования при разработке программного обеспечения позволяет организовать процесс сохранения объектов из оперативной в долговременную память. Данная статья посвящена описанию языка XOQL (XML Object Query Language), который представляет собой объектный язык запросов и для описания синтаксиса использует XML. В статье проведен глубокий и всесторонний анализ имеющихся работ. Обилие большого количества примеров позволяет продемонстрировать различные имеющиеся на сегодняшний день языки.

В работе представлен один из возможных вариантов представления базовых синтаксических конструкций объектного языка запросов в виде XML-документов. Перед проектированием синтаксиса были выделены критерии оптимальности, которые подробно описаны в работе. Кроме базовых, описаны синтаксические расширения языка запросов и способы расширения собственными конструкциями. Представлена структура реализованного оптимального языка с описанием тегов, атрибутов и допустимых значений.

В конце статьи представлено множество примеров различных видов запросов, часто встречающихся на практике.

Ключевые слова: объектный язык запросов, корпоративные информационные системы, язык запросов, язык разметки, XML, базы данных.

Цитирование: Oleynik P.P. XOQL: Object Query Markup Language // Business Informatics. 2015. No. 2 (32). P. 30–38.

Литература

1. Graves M. Designing XML databases. Prentice Hall PTR, 2001. 688 p.
2. Aho A., Lam M., Sethi R., Ullman J. Compilers: Principles, techniques and tools. Addison Wesley, 2006. 1000 p.
3. Holzner S. Real World XML (2nd Edition). Peachpit Press, 2003. 1200 p.
4. Albahari J., Albahari B. C# 5.0 in a Nutshell, 5th Edition. The Definitive Reference. O'Reilly Media, 2012. 1064 p.
5. Perrone P.J., Venkata S.R., Chaganti K.R. Building Java enterprise systems with J2EE. Sams Publishing, 2000. 1536 p.
6. Wood K. Delphi developer's guide to XML, 2nd Edition. Wordware Publishing, 2003. 545 p.
7. Wang J. Oracle Database 11g Building Oracle XML DB Applications. Oracle Press, 2011. 416 p.
8. Mistry R., Misner S. Introducing Microsoft SQL Server 2014. Microsoft Press, 2014.
9. Mullins C. DB2 Developer's guide – A solutions-oriented approach to learning the foundation and capabilities of DB2 for z/OS, 6th Edition. IBM Press, 2012. 728 p.
10. SQL: 2003 specification. [Электронный ресурс]: http://www.wiscorp.com/sql_2003_standard.zip (дата обращения: 10.01.2015).
11. XQuery 3.0 use cases. W3C Working Group Note 08 April 2014. [Электронный ресурс]: http://www.w3.org/TR/xquery-30-use-cases/ (дата обращения: 10.012015).
12. Rubinger A.L., Burke B. Enterprise JavaBeans 3.1, 6th Edition. O'Reilly Media, 2010. 766 p.
13. Nadkarni P.M., Brandt C.A., Morse R., Matthews K., Sun K., Deshpande A.M., Gadagkar R., Cohen D.B., Miller P.L. Temporal query of attribute-value patient data: utilizing the constraints of clinical studies // International Journal of Medical Informatics. 2003. No. 70. P. 59–77.
14. Leonard A. Pro Hibernate and MongoDB. Apress, 2013. 384 p.
15. Fowler M. Patterns of enterprise application architecture (Addison-Wesley Signature Series). Addison-Wesley Professional, 2002. 560 p.
16. Cattell R.G., Barry D.K. The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers, 2000. 288 p.
17. Jordan D. C++ Object Databases: Programming with the ODMG standard. Addison-Wesley, 1998. 460 p.
18. Mathematical Markup Language (MathML) Version 2.0. Second Edition. [Электронный ресурс]: http://www.w3.org/TR/MathML2/chapter4.html#id.4.4.3 (дата обращения: 10.01.2015).
19. Grand M. Patterns in Java. Volume 1. A catalog of reusable design patterns illustrated with UML. John Wiley & Sons, 1998. 480 p.
20. XSQL – Combining XML and SQL. [Электронный ресурс]: http://xsql.sourceforge.net/manual.php (дата обращения: 10.01.2015).
21. Collaborative Application Markup Language (CAML) structure specification. [Электронный ресурс]: http://download.microsoft.com/download/8/5/8/858F2155-D48D-4C68-9205-29460FD7698F/%5BMS-WSSCAML%5D.PDF (дата обращения: 10.01.2015).
22. Fox S., Johnson C., Follette D. Beginning SharePoint 2013 Development. Wrox, 2013. 456 p.
23. KDDML: A middleware language and system for knowledge discovery in databases. [Электронный ресурс]: http://kdd.di.unipi.it/kddml/papers/kddml.pdf (дата обращения: 10.01.2015).
24. KDDML language: Reference guide. [Электронный ресурс]: http://kdd.di.unipi.it/kddml/downloads/documentazione/Specifiche/kddml_specification_2_0_16.pdf (дата обращения: 10.01.2015).
25. Xu Y., Shi M. SQL Markup language for enterprise integration. Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 2004), 15-18 September 2004, Shanghai, China. 2004. P. 413–416.
26. ZsqlML (Zenark's XML for SQL). [Электронный ресурс]: http://sourceforge.net/projects/zsqlml/ (дата обращения: 10.01.2015).
27. Butek R. Web services tip: Use polymorphism as an alternative to xsd:choice. [Электронный ресурс]: http://www.ibm.com/developerworks/webservices/library/ws-tip-xsdchoice.html (дата обращения: 10.01.2015).