A New Life for SQL SELECT Statement

Bogdan SAHLEAN, Niculae DAVIDESCU
Department of Management Information Systems
Academy of Economic Studies, Bucharest, România
bsahlean@ase.ro, nicolae_davidescu@yahoo.com

An important percent from information systems use databases and in the majority of cases for developing such systems are used object oriented programming languages. From this point of view a key aspect is represented by the database querying features. The authors has observed a major gap between querying features of persistence mechanisms and the requirements for developing true object oriented software applications. Consequently, authors propose a new syntax for SQL SELECT statement, syntax that will allow to client applications to retrieve objects graphs.

Keywords: *object-oriented database, query, objects graph, SQL, syntax.*

Introduction

An important percent from software market is represented by software for enterprise. This software category can include software for manufacturing, warehouse management, supply chain, accounting, financials, human resources, decision support system and projects management. A study made by IDC[i] named "Worldwide ERP Applications Market 2006–2010 Forecast and 2005 Vendor Shares" shows "The ERP applications market grew 6.5% to $28.3 billion in 2005". Also, Albert Pang, Enterprise Applications Research Director at IDC, considers "SMB growth is expected to be phenomenal in the coming years as many of these ERP applications vendors released products that will be easier to implement with more preconfigured templates as well as hosting and Web services capabilities already built in". Conclusion is simple: enterprise software market is huge and it will continue to develop.

The main aspects concerning enterprise software development:

- Client – server architecture.
- For data storage are used databases (mostly relational databases).
- For implementing such applications are used in mostly cases object – oriented programming languages: Java, C#, C++, VB. TIOBE Programming Community Index for March 2008 shows a growing importance of object – oriented programming languages with +3.0 % compared with March 2007. In addition, this study shows that majority (54.9%) of programming languages are object – oriented.

Regarding to aspects presented, database connectivity from the point of view of object – oriented programming languages it is a subject what needs a special attention.

In this context, the keyword is persistence. The persistence must be defined in correlation with two objects – oriented concepts: class and object. Thus, Hans-Erik Eriksson and Magnus Penker propose the next definition: "a persistent class in one whose objects exist after the program that created it has existed […] Persistent class objects store themselves into a database, a file, or some other permanent storage" (Eriksson & Penker et al., 2003: 97). For Craig Larman, the persistence represents "The enduring storage of the state of an object." (Larman, 2002: 618).

Unfortunately this author's show only the first aspect of persistence: object state saving and ignore the second aspect that has the same importance: state loading.

In other words, persistence represent in the same measure:

- Object state saving (contents of its instance attributes) using a permanent storage device. This operation is represented by three services: C (create), U (update) and D (delete) from CRUD expression.
- Object state loading. This operation is represented by R (read) service.

At this moment, the most important solutions for persistence are:

- Object – oriented databases (OODBMS[ii]).
- Object – relational databases (ORDBMS[iii]).
- Relational databases (RDBMS[iv]) plus object – relational transformation (O/RM[v] mechanism).

Scott Ambler defines one of the most important characteristic for persistence mechanisms, characteristic that is query capabilities: "3. Multi-object actions. Because it is common to retrieve several objects at once, perhaps for a report or as the result of a customized search, a robust persistence layer must be able to support the retrieval of many objects simultaneously"(Ambler, 2005: 7).

Thus, for OODBMS, ORDBMS and RDBMS+O/RM, R (read) service must have two aspects:

- Defines possibility to restore object state (object loading) and
- Defines possibility to query objects.

From the point of view of query capabilities, we believe that persistence solutions must offer possibilities for query objects:

- To obtain a collection with objects of same type.
- To obtain a graph of objects from different classes (not one dimension table with records like RDBMS) at once.
- To obtain derived objects.

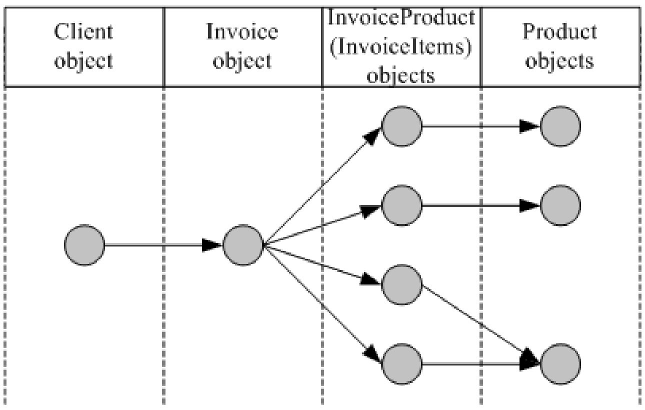


Fig.1. Object graph example

1. Object query capabilities: present

SQL is a standard query language that has been designed initial (SQL86 standard, SQL/1) for relational databases.

SQL99 standard (also named SQL/3) was the first step (in SQL standardization) from relation model to object model by including[vi] some elements from object model, elements named object – relational extensions, for examples:

- Table hierarchies (SQL1999 supports only single inheritance) and table types (tables defined based on UDT[vii]).
- User-defined data types:
  - New attributes and methods can be added.
  - Methods can be overridden.
- Reference types (REF).

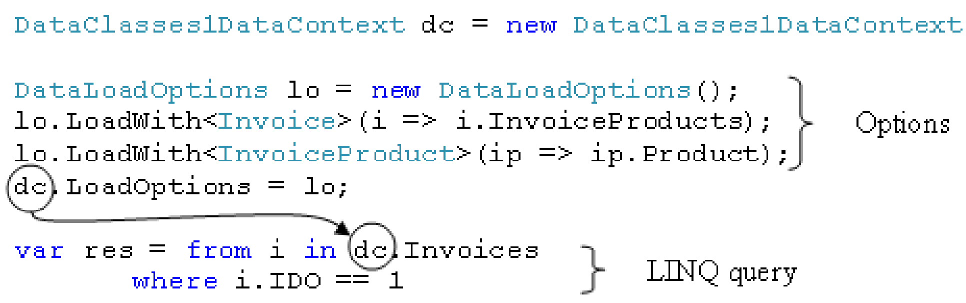
SELECT clause (from SQL SELECT statement) syntax has changed to permit calling methods using type.method(parameters) syntax.

From the point of view of object - oriented functionalities, SQL2003 and SQL2006 standards don't have anything new. Nevertheless, even if SQL1999 has introduced object – relational extensions in SQL standard, SELECT clause syntax has remained records oriented without allowing to select a graph of objects (like we have said in above chapter).

Other efforts have been made by ODMG[viii] to develop a query language for object – oriented databases. In 2001, has been adopted ODMG 3 standard that is the last standard. This standard proposes a new query language named OQL[ix]. Although OQL is like SQL, OQL does not have the same success.

Now, standardization effort is carried forward by Object Database Technology Work-

ing Group from OMG[x] that propose AOQL[xi]. Unfortunately, nor OQL and nor AOQL does not offer the possibility to obtain a graph of objects from different classes.
LINQ is at the same time a Microsoft .Net 3.5 Framework component but also a possible extension for programming languages for this platform but LINQ has been implemented for the first time as a library for .NET Framework 2. LINQ is a query language and it offers the possibility of querying a vast type of data sources including objects collection. LINQ syntax is derived from SQL SELECT statement syntax and includes Select operator to perform a projection. The biggest LINQ disadvantage (disadvantage derived from Select operator syntax) is the impossibility to obtain a graph of objects (see above, chapter 1).
Nevertheless, LINQ to SQL (not LINQ) persistence manager offers a solution: the programmers can use „eager loading" option, which is opposable to „lazy loading".
An example: let us assume Invoice, InvoiceProduct (or invoice item) and Product classes. If we want to load invoice object with ID = 1 with all items and products objects from a SQL Server database we must use load options:



```
DataClasses1DataContext dc = new DataClasses1DataContext();

DataLoadOptions lo = new DataLoadOptions();
lo.LoadWith<Invoice>(i => i.InvoiceProducts);
lo.LoadWith<InvoiceProduct>(ip => ip.Product);
dc.LoadOptions = lo;

var res = from i in dc.Invoices
        where i.IDO == 1
```

Fig.2. LINQ to SQL example

We can see that solution for loading Invoice object with all InvoiceProduct and Product objects is not a LINQ (query) solution but a LINQ to SQL (persistence manager) solution.

2. Object query capabilities: a proposal

In this paper, authors propose a new syntax for SELECT and FROM clauses from SQL SELECT statement to simplify objects graph loading.
Proposed syntax[xii] (*this syntax does not want to be a complete syntax*; authors wish to present only their contributions):
<select_statement> ::= <select_clause>
    <from_clause>
    <where_clause>
    <groupby_clause>
    <orderby_clause>
<select_clause> ::= SELECT OBJECTS <classes_list>
<classes_list> ::= <class>
  [PATH <class> [ON <relationships_list>][xiii]] …
<class> ::= <persistent_class> | <derived _class>
<persistent_class> ::= <class_identifier>
<derived_class> ::= <defined_derived_class> | <undefined_derived_class>[xiv]
<defined_derived_class> ::= NEW <class_identifier> ( )
<undefined_derived_class> ::=
  NEW OBJECT <class_identifier> (<undefined_derived_class_members>)
<from_clause> ::= FROM <classes_list>
<relationships_list> ::= <relationship>
    [AND | OR <relationship> …]
<relationship> ::= <class>.<collection_relationship>
    | <class>.<class_relationship>
    | <sql_condition>
Arguments:
<select_statement>
  Specifies type of the objects (classes) to be returned by the query. Classes used in SELECT clause can be persistent classes (classes that have objects stored in databases) and non-persistent derived classes

(classes that do not have objects stored in databases).

SELECT OBJECTS

Specifies that database engine must return objects, not plain records.

NEW <class_identifier> ( )

Specifies that the database engine must return non-persistent defined objects. These objects must have a class definition in database (much more like views in relational databases).

NEW OBJECT <class_identifier> (<untyped_derived_class_members>)

Specifies that the database engine must return non-persistent undefined objects. These objects do not have a class definition.

PATH

In SELECT clause, it is used to define objects graph. In FROM clause, it is used to define joins between source classes.

ON <relationships_list>

Specifies the condition on which the join or objects graph selection is based.

<from_clause>

Specifies the class(es) from which to retrieve objects.

Examples: let us assume that we have this classes:

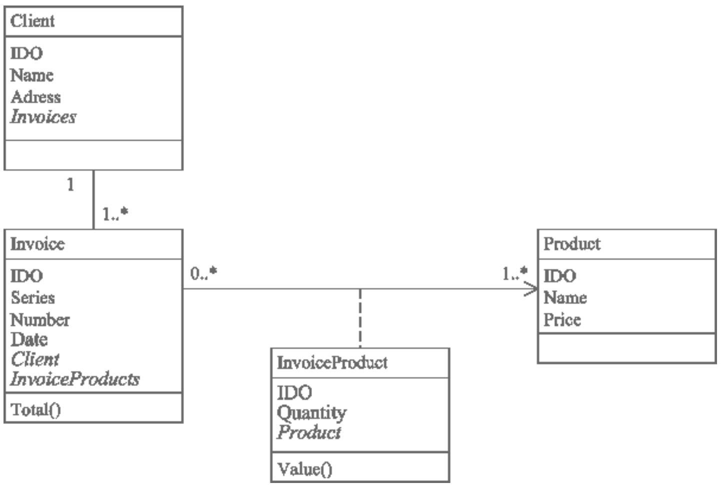


Fig3. UML class diagram

The following examples loads an invoice object (IDO = 1) and the correspondent client object.

SELECT OBJECTS Client PATH Invoice
FROM Client PATH Invoice ON Client.Invoices
WHERE Invoice.IDO = 1

Or

SELECT OBJECTS Client PATH Invoice
FROM Client PATH Invoice ON Invoice.Client
WHERE Invoice.IDO = 1

Or

SELECT OBJECTS Client PATH Invoice
FROM Client PATH Invoice ON Client.Invoices OR Invoice.Client
WHERE Invoice.IDO = 1

In the first solution, database engine is forced to join objects from Client and Invoice classes only using Client.Invoices relationship (we exclude Invoice.Client relationship). Because of that, it is possible that database engine not to find the optimum execution plan. This observation can be applied also for the second solution. The last solution is the most flexible and permits to obtain optimum execution plan.

The following example loads an invoice object (IDO = 1) with Client, all InvoiceProduct and all Product objects.

SELECT OBJECTS Client
    PATH Invoice
    PATH InvoiceProduct
    PATH Product
FROM Client
    PATH Invoice ON Client.Invoices OR Invoice.Client
    PATH InvoiceProduct ON In-

```
voice.InvoiceProducts
        PATH Product ON InvoicePro-
duct.Product
WHERE Invoice.IDO = 1
```

In this case, an objects graph is loaded from database[xv].

For an imaginary report created to print an invoice with IDO = 1 we can use the next query:

```
SELECT OBJECTS
NEW OBJECT InvoiceHeader /*One ob-
ject*/
( Series, Number, Date, Client.Name, Ad-
dress, Invoice.Total() )
PATH
NEW OBJECT InvoiceItem /*Many ob-
jects*/
( Product.Name, Quantity, Price, InvoicePro-
duct.Value() )
 FROM Client
PATH Invoice ON Client.Invoices OR In-
voice.Client
PATH InvoiceProduct ON In-
voice.InvoiceProducts
PATH Product ON InvoiceProduct.Product
WHERE Invoice.IDO = 1
```

Conclusions

We consider that this new syntax for SELECT and FROM clauses permits to obtain a graph of objects, thus simplifying development of object-oriented software.

References

[1] Ambler, S.W. (2005) „The Design of a Robust Persistence Layer For Relational Databases”, available on-line at http://www.ambysoft.com/downloads/persistenceLayer.pdf

[2] ANSI/ISO/IEC (2003) „SQL 2003 Standard”, available on-line at http://www.wiscorp.com/sql_2003_standard.zip

[3] Cattell, R., Douglas, K.B., Berler, M., Eastman, F., Jordan, D., Russell, C., Schadow, O., Stanienda, T. & Velez, F. (2000) *The Object Data Management Standard: ODMG 3.0*, Morgan Kaufmann

[4] Eriksson, H.E., Penker, M., Lyons, B. & Fado, D. (2003) *UML 2 Toolkit*, Wiley

[5] Gorman, M. (2001) „Is SQL Really A Standard Anymore?”, available on-line at http://www.tdan.com/view-articles/4923/

[6] Greene, R. (2006) „OODBMS Architectures”, available on-line at http://www.odbms.org/download/028.01%20Greene%20OODBMS%20Architectures%20September%202006.PDF

[7] Jeffrey, M.B. (2007) „Object-Relational Mapping as a Persistence Mechanism for Object-Oriented Applications”, available on-line at http://digitalcommons.macalester.edu/mathcs_honors/6/

[8] Larman, C. (2004) *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Unified Process*, Prentice Hall

[9] O_2 Technology (1998) „ODMG OQL User Manual”, available on-line at http://www.makumba.org/makumba/doc/oql-manual.pdf

[10] Pialorsi, P., Russo, M. (2007) *Introducing Microsoft LINQ*, Microsoft Press

[11] Taylor, A.D. (1998) *Object Technology*, Addison Wesley

[12] Tiobe Software (2008) „TIOBE Programming Community Index for March 2008”, available on-line at http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[i] International Data Corporation
[ii] Object Database Management System
[iii] Object Relational Database Management System
[iv] Relational Database Management System
[v] Object / Relational Mapping
[vi] in SQL standard
[vii] User-Defined data Types
[viii] Object Data Management Group
[ix] Object Query Language
[x] Object Management Group
[xi] Abstract Object Query Language
[xii] BNF syntax
[xiii] Optional for SELECT clause. In this case, the relationships list in SELECT clause will be the same like relationships list in FROM clause.
[xiv] Only in SELECT clause
[xv] See Figure 1