Hibernate the Recursive Queries - Defining the Recursive Queries Using Hibernate ORM

Aneta Szumowska, Aleksandra Boniewicz, Marta Burzańska, Piotr Wiśniewski*

Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Toruń, Poland

Abstract. The following paper presents results of combining two intensively developed technologies used to build database applications - SQL's recursive queries and object-relational mapping. Recursive queries are efficient tools for querying graph and hierarchical structures. They are very useful in solving problems such as searching for connections between two cities or calculating bill-of-material. Object-relational mapping allows for separation of business logic from database layer and more efficient implementation of computer software. However, modern mapping systems still lack support for many advanced SQL techniques. In order to perform a recursive calculation a programmer has to either write a recursive function in a language supported by the target database or implement a native function in the main software code that recursively sends queries to the database. The authors propose an enhancement enabling the use of recursive queries in one of the most popular ORM systems - Hibernate for Java language. With this enhancement a programmer will be able to take a full advantage of the support for the recursive queries offered by various object-relational database management systems and write a code fully compliant with Hibernate standard. The proposed solution works with IBM DB2, Oracle and PostgreSQL DBMS and proved to be many times faster than the approaches currently used.

1 Introduction

Recursion is one of the fundamental concepts of the computer science. It may be applied to problems dealing with corporate hierarchy, bill-of-material, travel connections or study classes organization. Data representing those problems is sometimes called *recursive data* because of its close relation with recursion. Vendors of Database Management Systems (DBMS) have long realized that recursive data need a separate approach. To meet the needs of object-relational database users SQL:99 standard introduced a special type of queries, whose computation was based on recursion. Those were recursive views and recursive common table expressions (RCTE). Even before the introduction of this standard, some database management systems supported proprietary recursive query constructs.

* emails: {iriz,grusia,quintria, pikonrad}@mat.uni.torun.pl

More about history of recursive queries, their availability and efficiency of evaluation in modern DBMSs may be found in [1, 2]. Despite the fact that recursive queries appeared quite a long time ago, intensive research is still being conducted on their optimization [3–5]. The time that has passed allowed the field of recursive queries to mature and now they become increasingly popular among software developers.

Parallel to the development of relational databases, the field of object-oriented programming languages and data modeling methods evolved as well. Modeling of relational data structures resembles more and more the modeling process of objects and classes. The issue has inspired the research on techniques of mapping objects to relations and relations to objects [6, 7]. The data transformations required in such mappings can be very complex, especially since they often involve advanced joins, nested queries and support for data update operations.

Object Relational Mapping tools allow the programmer to focus on the code development without the need for advanced knowledge of SQL nuances. This results in reduction of the time needed to develop software and an increased readability of the source code. Additional benefit of using object-relational mapping tools is the increased source code maintainability and portability between different DBMSs. The number of supported DBMSs depends on the actual ORM framework used - from one particular vendor to a full support of all major DBMSs.

Nowadays object-relational mapping tools are available for most of the popular programming languages. Due to the popularity of their base languages the most noteworthy are Hibernate and JDO for Java [8], LinQ and ADO .NET for .NET platform and SQLObject for Python. Because nowadays Java is the most popular language for software development we will restrict our discussion to it.

So far, recursive queries and ORM systems were two separate worlds. In order to query recursive data structures, the Java programmers had three alternative approaches to choose from. Firstly they could write a recursive stored function in a language supported by the target database. Secondly they could write an SQL query and send it directly through the ODBC/JDBC connections. The last possibility would be to write a recursive function in Java. Each of those solutions results in a number of problems. First two approaches result in a decreased maintainability, mainly due to the departure from the requirements of the ORM philosophy. The usage of ODBC/JDBC driver enforces implementation of additional safeguards from data types mismatch and unwanted side effects. On the other hand a function that recursively sends basic queries to the database in order to gather data generates unnecessary traffic which may have a negative impact on the general functionality of the application. It also lacks possible benefits from optimization of recursion that could be performed by the DBMS.

The first connection between ORM systems and a recursive queries technique has been made for a SQLObject - an ORM system for Python language, and a PostgreSQL DBMS [10].

After careful analysis of the current trends in software development, especially in the usage of ORM frameworks, the authors of this paper decided to base

their research on the Hibernate ORM [8, 11, 12]. Among the reasons behind this decision is Hibernate's current stage of development and wide-spread popularity. This tool is used in many projects, and although it was originally designed for Java language, it has been ported to other platforms including C#/.Net. Another important reason is that Hibernate is an open-source tool which enabled full integration of the proposed extension. Although Hibernate can be used with many different programming languages, the authors decided to implement discussed extension in Java language. However, this solution can easily be ported to other object programming languages.

2 Contribution

The algorithms for generating recursive queries and its implementation for IBM DB2, Oracle and PostgreSQL database management systems are the main contribution of the following work. An algorithm for generating recursive query for PostgreSQL widely extends the algorithm presented in [10]. The difference between them will be presented in *Features* section. Because of the differences between SQL dialects supported by mentioned database systems and the SQL:99 standard the authors had to carefully analyze different scenarios and establish standardized query generation methodologies.

The second equally important result is the development and implementation of interfaces for recursive query definitions for Hibernate ORM. These interfaces have been designed to retain maximum compatibility with standard mapping interfaces used in Hibernate. The key issue is to design them according to the Hibernate philosophy of the XML documents storing configuration data.

3 Hierarchical and Graph Data Structures

Development of computer science allowed for computer representation of mathematical structures such as graphs and trees. It soon became obvious that due to such representation many practical problems became easily solvable.

Well known and thoroughly discussed problems based on such structures deal with finding a path between two nodes, for example finding the communications links between the two cities or finding routes based on information provided by GPS systems. A particular type of graphs, called the trees, proved to be a very convenient scheme for hierarchies representation. Hierarchical data may be found at every step in everyday life. Every man is a part of their family tree. People working in large companies are part of the corporate hierarchy. Hierarchical data with a different nature are scoreboards at sporting championships. In graph and hierarchical data individual cells of information are linked together forming dependencies, which usually require recursive algorithms for analysis. Therefore, in this paper graph and hierarchical structures are commonly called the recursive structures.

The following section presents two natural examples of recursive data. Example 1 present data concerning corporate hierarchy with special focus on employees hierarchy. Example 2 describes a network of flight connections between cites.

The example data are presented tables 1 and 2

Table 1. Hierarchical data, table Emp

empId	**bossId**	**sname**	**fname**	**salary**
7369	7902	Green	John	100
7499	7698	Brown	George	210
7521	7698	Christie	Andrew	210
7566	7839	Jones	Brandon	360
7654	7698	Ford	Carl	210
7698	7839	Blake	Ernest	360
7782	7839	Bell	Gordon	360
7788	7839	Willis	James	360
7839		Smith	John	500
7844	7698	Turner	Johnathan	210
7902	7698	Adams	Trevor	210
7900	7566	Miller	Kyle	150

Table 2. Graph data, table Conns

departure	**arrival**	**flightId**	**price**	**travelTime**
Paris	Phoenix	TW 123	120	6h 15min
Paris	Houston	TW 120	130	8h 30min
Phoenix	Huston	PW 230	100	3h 10min
Huston	Chicago	RW 121	90	2h 45min
Huston	Dallas	RW 122	80	3h
Dallas	Chicago	DW 80	110	2h 30min
Chicago	Atlanta	CH 542	220	2h 45min
Chicago	Berlin	CH 543	360	7h 15min
Paris	Berlin	TW 118	300	1h 10min
Dallas	Berlin	DW 90	350	5h 45 min
Berlin	Boston	YW 421	100	6h
Chicago	Boston	CH 544	250	2h 15min

4 Data Representation in Object-Relational Mapping

Object-relational mapping systems are the result of intersection of two worlds - object-oriented programming languages and relational databases. They allow for separation of the business logic layer from database layer and as a result -

increased portability and maintainability of software's source code. Additional feature of ORM frameworks which has a big impact on their popularity is the speed of source code developments during which only minimal knowledge is needed about the advanced aspects of SQL language. There are many kinds of object relational mapping tools currently available on the market. They differ among themselves as to the programming language they are designed for and the scope of supported databases management systems. Some of them are available as commercial products, while others have a fully open source code. One of the most popular ORM frameworks is Hibernate for Java language.

Hibernate supports most of the major relational DBMSs. In accordance with JPA [9] standard it offers two methods of defining an object-relational mapping. To map Java classes to database tables, developer may choose to define a mapping configuration in an XML document or to define it using Java Annotations. Most of the programmers decide to use XML files choosing automatic generation of the corresponding Java classes performed by Hibernate. Than presented solution correspond to this method. Sample objects generated using Hibernate framework may resemble the classes form listings 1.1, 1.2.

Listing 1.1. Emp class representation

```
public class Emp {
  public long empId;
  public long bossId;
  public String sname;
  public String fname;
  public long salary;
  ...
```

Listing 1.2. Connections class representation

```
public class Connections {
  public String departure;
  public String arrival;
  public String flightId;
  public String price;
  public double travelTime;
  ...
```

Mapping system defines how the objects of those classes are mapped onto corresponding database tables. However, recursive querying of such defined data may case some problems. Source code written in Java that accomplishes this task is presented in listing 1.3. This is a straightforward code, but its evaluation time is not acceptable.

Listing 1.3. Recursive data retrieval using traditional tools

```
List<Empl> first = session.createCriteria(Empl.class).
    add(Restrictions.eq("sname", "Smith")).list();
Empl firstEmp = first.get(0);
```

```
while(!stack.isEmpty()) {
  Empl emp = (Empl) stack.firstElement();
  List<Empl> emps = session.createCriteria(Empl.class).
    add(Restrictions.eq("bossId", emp.getId())).list();
  for(int i = 0; i < emps.size(); i++)
    stack.push(emps.get(i));
  stack.remove(0);
}
```

The problem with long evaluation times comes from the fact that for each returned object, database server has to check if there are object with "subordinate" relation to that object. As a result, this query is being sent to the database as many times as there are employees in a sought structure. The solution proposed in this paper tested on a sample of 900 employees completed the evaluation process more than 20-times faster. The observed increase in evaluation speed has been achieved due to the usage of SQL's query type called recursive common table expression or in short recursive query.

5 Recursive Query

When browsing through recursive structures in a database management system that supports recursive queries, a user may use a special query type - a recursive common table expression. Such expression adjusted to query for Smith's direct and indirect subordinates is presented on listing 1.4

Listing 1.4. List of Smith's Subordinates

```
WITH RECURSIVE rcte (
  SELECT sname, fname, empId, False as isSub
    FROM Emp
    WHERE sname = 'Smith'
 UNION
  SELECT e.sname, e.fname, e.empId, True as isSub
    FROM Emp e JOIN rcte r ON (e.bossId = r.empId)
)
SELECT sname, fname
  FROM rcte
  WHERE rcte.isSub = True
```

Each recursive CTE consists of three parts: seed query, recursive query comprising references to the CTE being defined and an outer query that utilizes recursively generated data. Each of those queries may have their own selection predicates and have different tables declared in the FROM clause. More information on recursive SQL queries may be found in [1, 2].

6 Recursive Query in Hibenate

The authors have developed recursive query generators with automatic mapping of results onto objects. The author also provide interfaces for these generators. Listing 1.5 presents a configuration defined in XML document which allows for recursive querying of employees hierarchy. Using this configuration a programmer would gain access to the same set of results as when directly running the query from Listing 1.3

Listing 1.5. Mapping configuration for employees hierarchy

```
<rcte>
  <rcteTable name="subordinates"  max-level="4"/>
  <tables>
    <table>Emp</table>
  </tables>
  <recursive-condition>
    <on>Emp.bossId</on>
    <to>Emp.empId</to>
  </recursive-condition>
  <summands>
    <conc>Emp.empId</conc>
    <conc>Emp.sname</conc>
  </summand>
  <filter section="seed">
    Emp.sname = $Param(sname)
  </filter>
</rcte>
```

Just like basic Hibernate configurations our XML documents are used to generate special classes and objects that provide access to the specified set of results. Listing 1.6 presents an example of usage of object generated this way. This code gathers information about Smith's subordinates. The usage of newly generated objects is the same as standard objects generated by Hibernate thus it will not be discussed here.

Listing 1.6. Usage of rcte

```
try {
        SessionFactory session = HibernateUtil.getSession();
        Session s = session.openSession();
        Entity e = s.prepareRcte("subordinate");
        List<Map> rcteList = e.getRecursiveObjects();
        Iterator<Map> iter = rcteList.iterator();
        while (iter.hasNext()) {
                Map obj = iter.next();
                System.out.println(obj);
        }
        s.close();
```

```
} catch (Exception e) {
        e.printStackTrace();
}
```

7 Features

Configuration of a given query should specify the tables used to collect data. It also should comprise the joining predicates for those tables and, in particular, recursion predicate. Besides those information the construction the authors propose includes parameters helpful in specifying additional options. Listing 1.7 presents an XML configuration document for travel connections problem. It contains most of the parameter nodes available for the programmer.

The root of the configuration document is called `rcte`. Its first child node is used to specify the name of the output RCTE table (lines 2 and 3 of provided listing). This node is called `rcteTable` and has up to three attributes: `name` used to define the actual name of the RCTE, `max-level` specifying recursion depth and an optional `cycle` attribute used to enable cycle protection if it is provided by DBMS vendor. The next node of the configuration document is called `tables`. Its child nodes called `table` are used to specify which database tables will be used to generate the result. Names of those tables are passed as character elements. In the following example the only table needed to generate the result is Connections. It is represented by `<table>Connections</table>` code. As mentioned earlier, besides defining source tables, the programmer should supply information which table columns will be used in a recursion predicate. This information is stored in the `recursive-condition` node. This node has two child nodes: `on` and `to` which use character data to store corresponding column names. In the provided listing those nodes are placed in lines 7-10.

In addition to the required nodes describing source tables, developer may choose to specify additional options. Those include fields used to collect data such as sums or concatenations. The main node for such information is `summands`. It allows for specification of unlimited amount of two types of child nodes: `sum` and `conc`. The `conc` node is used to specify column which will be used to create a concatenated string. Concatenated values are separated by default using single white space. However, a programmer may choose to specify optional attribute `using` of the `conc` node to overwrite this character with a chosen string. Besides collection fields a programmer may also specify a constant field if needed. An example of such definition is `<const>Connections.arrival</const>` (line 17 of the listing 1.7). This field may be used by the presented generator to optimize resulting query using predicate push in technique described in [5].

To specify additional filtering predicates a programmer may use `filter` nodes. Depending on the target subquery this node may have three values if it `section` attribute: *seed*, *recursive* and *outer*. The programmer provides given predicates as character elements using *$Param()* function that enables passing of parameters in the Java source code. One last node supported by the generator is the `outer` node. Its contents define additional properties of the outer SQL

query that uses the rcte. This node may additionally contain `property` tags, which have the same meaning as identical elements in classical Hibernate configuration files. For example, they may be used to select only relevant columns instead of all generated ones.

Listing 1.7. Mapping configuration for travel connections

```
<rcte>
   <rcteTable name="travel" max-level="4" cycle="false"/>
   <tables>
     <table>Connections</table>
   </tables>
   <recursive-condition>
      <on>Connections.departure</on>
      <to>Connections.arrival</to>
   </recursive-condition>
   <summands>
     <sum>Connections.travelTime</sum>
     <conc>Connections.flightId</conc>
     <conc using=";">Connections.arrival</conc>
   </summands>
   <constants>
     <const>Connections.arrival</const>
   </constants>
   <filter section="seed">
     Connections.departure = $Param(departure)
   </filter>
   <filter section="outer">
    Connections.arrival = $Param(arrival)
   </filter>
</rcte>
```

At first glance, this configuration may seem very complex. However there are clearly distinguished sections corresponding to the elements described. Names of the configuration nodes have been chosen so that they would be self-explaining in most cases. Based on such configuration the query generator will construct a corresponding recursive SQL query. For the configuration presented in listing 1.7 the output query with already defined filtering parameters is about one page huge. This query shows the transparency of the configuration presented in listing 1.7 in comparison with a generated query.

The first algorithm for generating recursive queries presented by the authors in [10] has a series of constraints. First it works only with PostgreSQL. Second it works only with one source table. Third it has no filter clauses. Fourth it determines the names for summands, concats etc., then fields in generated class sometime have complicated names. The algorithms presented in this paper fulfill these features, thus they are much more useful than the previous one.

In the paper [2] authors have shown that the performance of recursive queries in DB2 database system highly depends on existing indexes placed on the fields

from recurrence predicate. Thus the authors of the following paper have decided to add a special attribute `indexed = True/False` in the recurrence condition definition. If it is set to `True` the configurator checks if the proper fields are indexed and if not - inserts missing indexes into the database. With accordance to [2] this attribute is ignored by configurator for database systems other than DB2.

8 Conclusions and future work

In conclusion, the results of the tests show that the solution proposed by the authors allows for benefiting from the advantages of Object Relational Mapping in applications using recursive data. As it has been described both presented configuration methods preserve the original Hibernate's configuration style which makes them easily accessible for the user.

Next the authors plan to continue the research on providing support for another features implemented in database management systems at the level of Object Relational Mapping.

References

1. Brandon, D. 2005. Recursive database structures. J. Comput. Small Coll. 21, 2 (Dec. 2005), 295-304
2. Boniewicz A., Burzanska M., Przymus P.: Recursive query facilities in relational databases: a survey, In DTA(2010), CCIS series, Volume 118, 89-99
3. Ghazal, A., Crolotte, A., Seid, D.Y.: Recursive SQL Query Optimization with k-Iteration Lookahead. In DEXA(2006) 348-357
4. Ordonez, C.: Optimization of Linear Recursive Queries in SQL. IEEE Trans. Knowl. Data Eng.(2010) 264-277
5. Burzańska, M., Stencel, K., Wiśniewski, P.: Pushing Predicates into Recursive SQL Common Table Expressions. In ADBIS(2009) 194-205.
6. Melnik, S., Adya, A., Bernstein, P. A.: Compiling mappings to bridge applications and databases. ACM Transactions on Database Systems (TODS), v.33 n.4, p.1-50, 2008
7. Keller, W.:Mapping objects to tables: A pattern language. In EuroPLoP (2007)
8. Hibernate, http://www.hibernate.org
9. DeMichiel, L.: Java Specification Requests JSR 317: JavaTM Persistence 2.0. http://jcp.org/en/jsr/detail?id=317, 2009.
10. Burzańska M., Stencel K., Suchomska P., Szumowska A., Wiśniewski P.: Recursive Queries Using Object Relational Mapping. FGIT(2010) LNCS, Volume 6485/2010, 42-50
11. Christian Bauer , Gavin King, Java Persistence with Hibernate, Manning Publications Co., Greenwich, CT, 2006
12. Elizabeth J. O'Neil. 2008. Object/relational mapping 2008: hibernate and the entity data model (edm). In Proc. ACM SIGMOD (2008), 1351-1356.