Performance Tuning of Object-Oriented Applications in Distributed Information Systems

Zahra Davar and Janusz R. Getta
School of Computer Science and Software Engineering, University of Wollongong, Wollongong, NSW, 2522, Australia

Keywords: Global Information System, Data Integration, Object-Oriented Application, Transformation Rule.

Abstract: Majority of the global information systems are constructed from a number of heterogeneous distributed database systems that provide a global object-oriented view of the data stored at the remote systems. Such global information systems have two sides: the source side which consists of heterogeneous distributed databases and the global side which provides an integrated view of the database systems from the source side. User applications access data through the iterations over the classes of objects included in a global object-oriented view. The iterations over the classes of objects are implemented as iterations over the data items such as the rows in the relational tables on the source side of the system creating serious performance problems.

This paper addresses the performance problem of object-oriented applications accessing data on the source side of global information system through an object-oriented view on a global side. We propose a number of transformation rules which allow for more efficient processing of object-oriented application on the source side. The rules can eliminate the iterations of classes of objects on the global schema side. We prove the correctness of the rules and show how to systematically apply the rule to object-oriented applications. The paper proposes a number of templates for programming of object-oriented applications that allow for easier and more efficient performance tuning transformations.

1 INTRODUCTION

Distributed information systems are becoming increasingly important and gaining a great deal of attention in commercial and business applications. Global information systems used by the large organisations have heterogeneous and distributed structures (Lopatenko, 2004).

In this paper, we consider a class of distributed information systems which consist of the distributed and homogeneous database systems on a source side and the global object-oriented view of data on the client side. The source database side includes relational database systems and the applications on a client side access data from a global object-oriented view of all distributed databases. Figure 1 illustrates the model of the distributed system. On the server side, there is some relational databases, e.g A and B. The arrows show that the relational databases will transfer from the source database side to the client side as classes of objects and then object-oriented developers can access data.

An object-oriented view of data on a global side of the system allows application programmers to access data through iterations over the classes of objects. Such approach to implementation of user applications reduces the amounts of nonprocedural code when accessing data , e.g. joins of relational tables in *SELECT* statements of SQL into nested loops iterating of the objects. It means that all of the data must be transferred from the source database side to an application accessing the global view. Therefore, the filtering conditions of the application will apply to all transferred data in the global view side. This leads to iterations over a large number of objects on the global view side, a procedure which is inefficient for large databases. In addition, to process data on the client side, the application uses some algorithms which are not as efficient as the algorithms which can process the same data on the server side. For instance, the implementation of *JOIN* operation on the server side is more efficient than implementation of the same *JOIN* operation on the client side by two nested loops.

Implementing efficient object relational applications for such systems is a serious challenge. There are different ways to solve this problem. One solu-

Davar Z. and R. Getta J..
Performance Tuning of Object-Oriented Applications in Distributed Information Systems.
DOI: 10.5220/0004887402010208
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 201-208
ISBN: 978-989-758-027-7

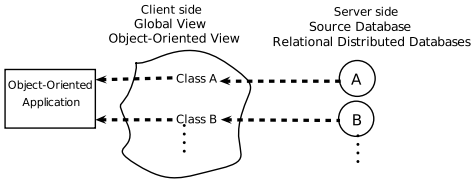

Figure 1: Model of the distributed system

tion is based on changing the object relational mapping. By default, object relational mapping shifts the data processing to the client side to be consistent with the virtual object database. This can be changed so that data processing shifts to the source database side. As a result, less data will transfer from the source database side to the client side. The other solution is to reduce the transmisison of a large number of data from the source database side to the global view side. This can be done by changing the configuration of the application with more non-procedural code. This means that, only the objects needed to satisfy the filtering conditions of the application will be transferred from the source database side to the global database.

This paper presents a number of transformation rules which can eliminate iteration over a large number of objects by changing the configuration of the application. To do this, we used more OQL in the body of the application. As a result faster and more efficient performance of the application is achieved.

In the remainder of this paper, first the motivation experiment is presented. Section 2, examine the current research. Input patterns of the rules for different styles of programs is presented in section 3. Section 4 presents the transformation rules. Verification and logical proof of the rules is in section 5. Section 6 contains the conclusion and suggested future work.

1.1 Motivation Experiments

In this section, motivation experiments are presented. These experiments were conducted on the TPC Benchmark website which has 300 MB relational data. All the experiments were run on a Dell system with 3.33GHz intel(R), Core(TM)2, Duo CPU by Ubuntu 10.04 LTS, the Lucid Lynx. The system had 3.25GB RAM. The examples were run in Java Persistence API (JPA) programming language format and in the NetBeans 7 environment. The netbeans 7 clock were used to monitor execution time by seconds.

Application 1, is the *loop* part of an original application based on JPA, written by an object-oriented programmer. Application 1 is an example of a program that iterates over two relational databases, called *Lineitem* and *Supplier*. This algorithm retrieves all values from *l.partsupp* in the *Lineitem* class which has equal value to the *s.suppkey* in the Supplier class. We managed various experiments for different sized databases, and measured the response time before and after transformation. In all examples, *Supplier* includes 3000 objects but size of the *Lineitem* class varied between 100,000 objects to 2,000,000 objects.

Application 1:

```
{
 Query query1 = em.createQuery
 ("SELECT  s FROM Supplier s
   ORDER BY s.sSuppkey", Supplier.class);
    List list1 = query1.getResultList();
    Iterator iterator1= list1.iterator();
      while(iterator1.hasNext()){
        Supplier s= (Supplier)iterator1.next();
        Long ps_suppkey = s.getSSuppkey();
  Query query2 = em.createQuery
  ("SELECT  l FROM Lineitem l
   WHERE l.partsupp.supplier.sSuppkey= "
    + ps_suppkey,Lineitem.class);
 List list2 = query2.getResultList();
 Iterator iterator2= list2.iterator();
 }
```

By changing the configuration of application 1, we were able to write application 2 which has exactly same output as application 1. In application 2, we obtained two *SELECT* statements together and used one *SELECT* statement. By using application 2 and changing the balance of the data processing, the unnecessary iteration is eliminated. Also, configuration of application 2 is such, that not all of the objects will transfer from the server side to the client side. As a result, the time needed for execution of the application was reduced.

Application 2:

```
{
Query query = (Query) em.createQuery
("SELECT l.partsupp.partsuppPK.psSuppkey,
COUNT(l.partsupp.partsuppPK)
FROM Lineitem l JOIN l.partsupp ps
GROUP BY l.partsupp.partsuppPK.psSuppkey
ORDER BY l.partsupp.partsuppPK.psSuppkey",
Lineitem.class);
      List<Object[]> list = query.getResultList();
      Iterator iterator1= list.iterator();
 }
```

Figure 2, illustrates the execution time which was needed to run application 1 with different sizes of the *Lineitem* class. We started running application 1 with 100,000 objects in relational database *Lineitem* and increased the objects up to 2,000,000. The results made an exponentially chart shown as Figure

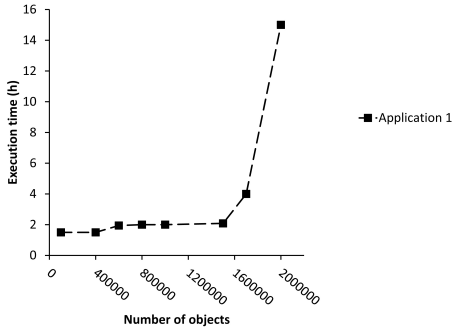

Figure 2: Execution time for application 1.

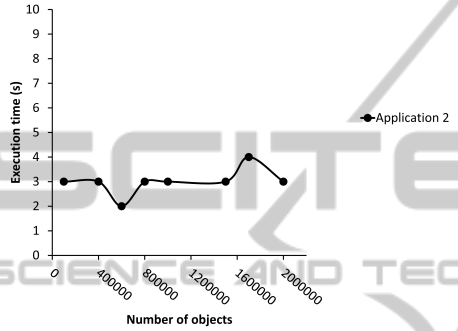

Figure 3: Execution time for application 2.

2. The response time starts from approximately 1.5 hours and increase to almost 15 hours.

Figure 3, illustrates the execution time of application 2 with same the data as we explained earlier. The runtime of application 2 varied between 2 to 4 seconds as shown on the linear graph in Figure 3. These results show that if we transmit 300MB instead of 3GB to the client side the output will be at least 100 times faster.

2 EXISTING WORKS

The existing works are divided into two categories. One is related to integrating data models into distributed database applications and the other is focused on performance evaluation of relational database applications. Regarding integrating data, (Mansuri and Sarawagi, 2006) proposed a system for integrating unstructured data into relational databases. The main context of their research in (Mansuri and Sarawagi, 2006) addresses the integration of unstructured text records into existing multi-relational databases. Another work focused on increasing automated integration using standardisation (Lawrence and Barker, 2001). A different approach based on the existence of virtual mining views was proposed to integrate patterns in relational databases (Calders et al., 2006). An approach to the process of integration of complex database using IIS*Case is proposed by (Lukovi et al., 2006). In all the above research, the performance problem of distibuted information systems in global schema side was not considered as a challenge in integrating the data models.

Regarding the performance problem of relational applications, (Agarwal, 1995) proposed the idea of using a client-side object cache to increase database application performance. Other works, e.g. (Van Zyl et al., 2006)(Kalantari and Bryant, 2012), focus on comparison of performance in relational applications. (Van Zyl et al., 2006) compared the performance of object-relational mapping and object database management system. In (Van Zyl et al., 2006), object-relational mapping in the open source applications were discussed. Based on the object's elaboration, object and object-relational database applications were compared in (Kalantari and Bryant, 2012). Similar views to ours has been done by other research (Rumbaugh et al., 1991), (Kroenke, 2001) and (Rahayu et al., 2001). In the research paper by Rahaya et al.(Rahayu et al., 2001), the performance of object-relational transformation methodology was compared with that of the ordinary relational method. In that research, object-relational transformation methodology is used to prove the efficiency of the operations on the relational tables. In the other works (Rumbaugh et al., 1991), (Kroenke, 2001), some rules were developed to transfer object-oriented design to relational tables. The gap in these studies (Rumbaugh et al., 1991), (Kroenke, 2001), (Rahayu et al., 2001) is that only the conceptual parts of object-orientation are considered and dynamic parts are not involved.

All this research is based on shifting the data-process to the client side rather than the server side. This means that the problem of the need for multiple unnecessary iterations remains in all of these approaches. What is needed is an approach which can eliminate these and thus provide much greater efficiency and much higher speed.

3 TRANSFORMATION RULES

By using the transformation rules presented in this paper, non-optimised versions of object-relational database applications can be optimised to provide the necessary efficiency and high speed. They replace fragments of procedural code with non-procedural statements which leave most of the data processing on the server side and this improves the performance of an application by eliminating inessential data processing. In our approach, the filtering con-

ditions are evaluated on the source schemas side (the server side) and as a result, there is no need to transmit many data to the client side. This can eliminate iterations over unnecessary classes of objects. Through our experimental results based on large scale relational databases, we show that our transformation rules eliminate inessential interaction between the server and the client side.

We now provide the transformation rules. In this paper we limit ourselves to the rules applicable only to the most widely used programs, although others have already been designed.

Based on our approach, relational applications are divided into the following categories:

- Iteration over two classes of objects/Join
- Anti-Join
- Aggregation

These rules can convert most of the applications because large applications are a mixture of the above applications. The rules are based on two parts: input and output components. Input component is an original object oriented program or part of that. The output components are transformed version of the input components which can run much faster that the input component. The design of the output components are based on using more OQL.The input components should be compatible with one or more of the templates in section 4.

3.1 Iterations over Two Classes of Objects

Algorithm 1 is the input component for rule 1. This algorithm includes two nested *SELECT* statements which perfomes *JOIN* operation. Algorithm 2, is the transformation rule which transforms the input component to the optimise version. This rule merges two *SELECT* statements in the body of the original program into one *SELECT* statement by using a *JOIN* clause. Based on the proposed approach, this rule changes the configuration of the original program by using more non-procedural code.
For an example of the application of this rule, see section1.1. Concatenation is used between non-relational conditions to put together all the results from both classes. By using this rule, only the objects which satisfy the *JOIN* condition will transfer from the source schema side to the global schema side. Therefore, for big databases it can save runtime as unneeded objects will remain on the source schema side.

Algorithm 1: **Input component.**
Iterations over two classes of objects.

```
1 for each t in (SELECT * FROM Class1
  WHERE φ [t1,...,tn]) do
2 |   for each s in (SELECT * FROM Class2
  |   WHERE γ [s,s2,...,sn] + γ'
  |   [< s1,t1 >,...,< sn,tn >]) do
3 |   |   Write t
4 |   |   Write s
5 |   end
6 end
```

Algorithm 2: **Output component.**
Rule 1/ Iterations over two classes of objects.

```
1 for each p in (SELECT * FROM Class1 JOIN
  Class2 on
2 γ' [< t1,s1 >,< t1,s2 >,...,< tn,sn >]
3 Where
4 φ [t1,...,tn] || γ [s1,s2,...,sn] do
5 |   Write p
6 end
```

3.2 Anti-join

Algorithm 3 is the input component for *anti-join* which is one of the possible implementation of *anti-join* for object-oriented developers. The output component, includes only one *SELECT* statement. In algorithm 4, the *left-outer-join* is used to select the nessesary data and transfer them to the client side.

Algorithm 3: **Intput component.**
Anti-Join.

```
1 for each t in (SELECT * FROM Class1) do
2 |   for each s in (SELECT count(*) FROM
  |   Class2 WHERE
  |   Class2.Memberj=t.Memberi) do
3 |   |   if Count=0 then
4 |   |   |   Write t
5 |   |   end
6 |   end
7 end
```

The following algorithm is considered as transferred version of the input components of *anti-join* applications.

3.3 Aggregation

Algorithm 6 is the rule for counting representative objects from a class of objects. The input component of this rule is presented as algorithm 5. The Group by clause is used to group the necessary objects and transfer them to the global schema side.

Algorithm 4: **Output component.**
Rule 2/Anti-Join by Outer join.

```
1  for each p in (SELECT * FROM Class1 LEFT
   OUTER join Class2 on Class2.Memberj =
   Class1.Memberi) do
2  |  if Class2.Memberj is Null then
3  |  |  Write p
4  |  end
5  end
```

Algorithm 5: **Input component.**
Counting objects.

```
1  ArrayList Memberi = new array list()
2  int counter= new int[ ]
3  for each t in (SELECT Memberi FROM Class1
   ) do
4  |  counter[getInt("t.Memberi")]++;
5  end
6  for i=0 ; i<t.Memberi.length; i++ do
7  |  if counter[i]>t.Memberi.length then
8  |  |  Write counter[i]
9  |  |  Write i
10 |  end
11 end
```

Algorithm 6: **Output component.**
Rule 3/ Counting objects.

```
1  for each t in (SELECT Memberi,COUNT(*)
   FROM Class1 GROUP BY Memberi) do
2  |  x = getInt(Memberi)
3  |  y = resultset(COUNT(*))
4  |  Write x
5  |  Write y
6  end
```

4 TEMPLATES FOR USING THE RULES

In this section, we present the styles which individual JPA programmers can use to write a relational application. The application developers can design applications in different ways and in this case we would need many different rules to cover all possible cases. To analyse the source code easier, we proposed certain patterns for input components of the rules. The transformation rules which are presented in section 3 can only apply to original programs which are consistent with the following styles. The styles are designed to match the most current object-oriented programs. Because the majority of relational database applications are combinations of the following styles, our transformation rules are applicable to them. The following styles are designed based on JPA programming language and are similar to most of the other object-oriented languages. Depending on the original program, the names of the objects and classes, the relational conditions and the non-relational conditions will change.

4.1 Style 1

- Iterations over two classes of objects Join

' t ' is set of objects from class1 and ' s ' is set of objects from class2.

Non relational conditions of class1 : φ $[t.t1, t.t2, ..., t.tn]$. Example: Class1.Objecti= X

Relational conditions : γ' $[< s1, t1 >, ..., < sn, tn >]$. Example: Class2.Objectj=Class1.Objecti

Non relational condition of class2 : γ $[s, s2, ..., sn]$. Example: Class2.Objectj=Y

```
{
    ResultSet rset1 = stmt1.executeQuery
    ("SELECT t FROM CLASS1");
    GET VARIABLE ;
    "NON RELATIONAL CONDITIONS of CLASS1";
    While (rset1.next() )
    {
      VARIABLE =  rset1.getInt(1);
      ResultSet rset2 = stmt2.executeQuery
      ("SELECT s FROM CLASS2
       WHERE
       "RELATIONAL CONDITIONS");
         While ( rset2.next() )
         {
     "NON RELATIONAL CONDITIONS of CLASS2";
         }
         System.out.println( VARIABLE );
}
}
```

If the input component matches this style, Rule 1 which is presented in 3.2 can modify the configuration of the application.

4.2 Style 2

- Anti-Join

For anti-join, there are two styles which are most commonly used by programmers.

```
{
    ResultSet rset1 = stmt1.executeQuery
    ("SELECT * FROM CLASS1");
```

```
    GET VARIABLE = FALSE ;
    While (rset1.next() )
    {
      ResultSet rset2 = stmt2.executeQuery
      ("SELECT * FROM CLASS2
       WHERE
       "Class2.OBJECTj = Class1.OBJECTi");
         While ( rset2.next() )
         {
     VARIABLE = True;
         Exit;
         }
        If VARIABLE=FALSE
        {
        System.out.println(OBJECTi);
        }
}
}
```

Another style for anti-join with counter:

```
{
    ResultSet rset1 = stmt1.executeQuery
    ("SELECT * FROM CLASS1");
    While (rset1.next() )
    {
      ResultSet rset2 = stmt2.executeQuery
      ("SELECT Count(*) FROM CLASS2
       WHERE
       "Class2.OBJECTj = Class1.OBJECTi");
         While ( rset2.next() )
         {
         If Count=0
     {
          System.out.println(OBJECTi);
        }
}
}
}
```

If the input component matches one of the anti-join styles presented here, then it can be modified according to Rule 2 which is presented in 3.2 .

4.3 Style 3

- Aggregation

For aggregation queries, the style for counting objects from a class and grouping the similar objects is presented. The styles for the rest of the aggregations are much the same as the following style so we do not mention them here.
If the original application be consistant with the following style, then it can be optimise by Rule 3 which is presented in 3.3.

```
{
    ArrayList Memberi = new array list();
    int counter= new int[ ];
    ResultSet rset1 = stmt1.executeQuery
    ("SELECT * FROM CLASS1);
     {
      While {ArrayList.next}
        {
        counter[getInt("t.Memberi")]++;
        }
      For i=0;i<t.Memberi.length;i++
      {
       System.out.println(MyCounter[i]+ i);
      }
}
}
```

There are more styles used by object-oriented applications, especially for anti-join and aggregations, but not all styles can be presented in this paper. The templates do however, cover most of the possible input components.

5 VERIFICATION OF THE RULES

The *Floyd Hoare logic* formula (Alagic and Arbib, 1978) is used to show the correctness of each rule by itself and also to show that each rule and its pattern can achieve the same output. Writing the same invariant for each rule and its pattern allows us to prove each rule. The algorithms under analysis were rewritten with *WHILE* in order to be consistent with the *Floyd Hoare logic* format (Alagic and Arbib, 1978). By writing an invariant for each algorithm, the correctness of each individual algorithm is proved and by verifying the same invariant for the pattern of the rules, the equality of both components is proved. In this section we describe the proof for one of the group of applications (iterating over a class of objects) along with the method of verification of the rule.

Algorithm 7, is an input component which is rewritten with *pseudo-code* by *WHILE* and includes the invariant. This pattern is designed for iterating over one class of object. As this is the simplest pattern in object-oriented applications, we describe the correctness of this pattern and its rule to show the method that used to prove the other patterns and their rules. In line 3, before entering the *WHILE* loop, there is a condition and an invariant that are true at that stage of the algorithm. The condition is 't $<>$ Nil' which means there should be at least one object in class 1 before entering the *WHILE* loop. This condition is true before entering the *WHILE* loop. The invariant before and after the *WHILE* loop is the same: '$\forall$ t $\in$ R : $\varphi[t]$'. This means only objects that satisfy the condition of $\varphi[t]$ will go into the output. This invariant is true before entering the *WHILE* loop and also after exiting or skipping the *WHILE* loop. In the body of the program, each object which satisfy condition $\varphi[t]$, will add to the result set R.

Assume that R is the results set

Algorithm 7: Input component.
Iterating over a class of object.

```
1  R:= Ø
2  Get (t,C)
3  (t <> Nil) ∧ ∀ t ∈ R : φ[t]
4  while t <> Nil do
5  |  if φ [t] then
6  |  |  R:= R∪ t
7  |  |  Get (t,C)
8  |  end
9  end
10 ∀ t ∈ R : φ[t]
```

Algorithm 8: Output component.
Rule for iteration over a class of object.

```
1  R:= 0
2  Get (t,C')
3  (t <> Nil) ∧ ∀ x ∈ C' : φ[x] ∧ ∀ t ∈ R : φ[t]
4  while t <> Nil do
5  |  R:= R∪t
6  |  Get (t,C')
7  end
8  ∀ t ∈ R : φ[t]
```

Assume that C is the: SELECT * FROM class 1
Assume that t is a set of objects from class 1
Assume that $\varphi\,[t.t1, t.t2, ..., t.tn] = \varphi\,[t]$
In the Floyd Hoare logic formula which is presented as equation (1):

P is the loop invariant which is to be preserved by the loop body S. B is conditions of the application. After the loop is finished, this invariant P still holds, and moreover ¬ B must have caused the loop to end (Alagic and Arbib, 1978). The logic includes two separate parts and by proving the first part, the second part is automatically proved. The following line is applied to the application as equation (2).

{Condition} ∧ {Invariant} [*Body*] {Invariant}

The equation 2 is always true as explained earlier. Therefore the second part of the logic is automaticaly true: $\{Invariant\}while(Condition)body\{\neg Condition \wedge Invariant\}$. This is presented as equation (3).

In this stage, the pattern is proved. The next stage is to prove that the rule is correct by applying the *Floyd Hoare logic* formula to the rule. Also, to make the assumption that each pattern and its rule are doing the same thing, we should achieve the same invariant from both patterns and their rules. Algorithm 8, is the rule for itterating over a class of object.

Assume that C' is : SELECT * FROM class 1 WHERE $\varphi\,[t1, t2, ..., tn]$

We apply the *Floyd Hoare logic* formula to the rule as equation (4).

This is always true. Before entering the *WHILE* loop, both the condition and the invariant are true because there must be at least one object to enter the *WHILE* loop and then each object t can satisfy the condition of $\varphi\,[t1, t2, ..., tn]$. After exit or skipping the loop, all the objects in set R can still satisfy the condition of $\varphi\,[t1, t2, ..., tn]$. Therefore, based on *Floyd Hoare logic*, this rule is proved (Alagic and Arbib, 1978). In addition, both the pattern and its rule include the same invariant, so this can prove the equality of both algorithms. The same method is used to show the correctness of the other patterns and their rules.

$$\frac{\{P \wedge B\}S\{P\}}{\{P\}while(B)do(S)\{\neg B \wedge P\}} \quad (1)$$

$\{(t <> Nil) \wedge \forall\, t \in R : \varphi[t]\ \}$[If $\varphi\,[t]$ Then R:= R∪ t , Get (t,C)] $\{\forall\, t \in R : \varphi[t]\}$ (2)

$\{\forall\, t \in R : \varphi[t]\}$ while (t <> Nil) [If $\varphi\,[t]$ Then R:= R∪ t , Get (t,C)] $\{$ (t : Nil) $\wedge \forall\, t \in R : \varphi[t]\ \}$ (3)

$\{(t <> Nil) \wedge \forall\, x \in C' : \varphi[x] \wedge \forall\, t \in R : \varphi[t]\ \}$ R:= R∪ t, Get (t,C') $\{\forall\, x \in C' : \varphi[x] \wedge \forall\, t \in R : \varphi[t]\ \}$ (4)

6 CONCLUSION AND FUTURE WORK

In this paper we attempt to solve the performance problem of object-oriented applications within a distributed information system. We introduced some transformation rules that can shift more data processing to the source schema side. Using this approach, increases the amount of non-procedural code as opposed to procedural code in the body of object- oriented applications. Our approach reduces iterations over classes of objects because only the necessary objects will transfer from the source schema side to the global schema side. Therefore, the proposed transformation rules lead to high performance relational applications. We considered three categories of styles for our transformation rules and prove the correctness of each rule based on Floyd Hoare logic. Templates of the rules were also presented. Designing more input patterns for the rules and also designing machinery to optimise object relational applications automatically remain for future work.

REFERENCES

Agarwal, S. (1995). Architecting object applications for high performance with relational databases. In *In OOPSLA Workshop on Object Database Behavior, Benchmarks, and Performance*.

Alagic, S. and Arbib, M. A. (1978). *The Design of Well-Structured and Correct Programs*. Springer.

Calders, T., Goethals, B., and Prado, A. B. (2006). Integrating pattern mining in relational databases. Springer.

Kalantari, R. and Bryant, C. H. (2012). Comparing the performance of object and object relational database systems on objects of varying complexity. In *Proceedings of the 27th British national conference on Data Security and Security Data*, pages 72–83, Berlin, Heidelberg. Springer-Verlag.

Kroenke, D. M. (2001). *Database Processing: Fundamentals, Design and Implementation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 8th edition.

Lawrence, R. and Barker, K. (2001). Integrating relational database schemas using a standardized dictionary. In *Proceedings of the 2001 ACM symposium on Applied computing*, SAC '01, New York, NY, USA. ACM.

Lopatenko, A. (2004). Query answering under exact view assumption in local as view data integration system. In McIlraith, S. and Morgenstern, L., editors, *Proceedings of the Doctorial Consortium 9th International Conference on Principles of Knowledge Representation and Reasoning*.

Lukovi, I., Risti, S., Mogin, P., and Pavievi, J. (2006). Database schema integration process a methodology and aspects of its applying. In *Sad Journal of Mathematics (Formerly Review of Research, Faculty of Science, Mathematic Series), Novi Sad, 2006, Accepted for publishing*.

Mansuri, I. R. and Sarawagi, S. (2006). Integrating unstructured data into relational databases. In Liu, L., Reuter, A., Whang, K.-Y., and Zhang, J., editors, *ICDE*, page 29. IEEE Computer Society.

Rahayu, J. W., Chang, E., Dillon, T. S., and Taniar, D. (2001). Performance evaluation of the object-relational transformation methodology. *Data Knowl. Eng.*, 38:265–300.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, USA.

Van Zyl, P., Kourie, D. G., and Boake, A. (2006). Comparing the performance of object databases and orm tools. In *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*. South African Institute for Computer Scientists and Information Technologists.