CHAPTER 3

AUTOMATIC CODE GENERATION FOR RECURRING CODE PATTERNS IN WEB BASED APPLICATIONS AND INCREASING EFFICIENCY OF DATA ACCESS CODE

3.1 INTRODUCTION

Today, a lot of web applications and web sites are data driven. These web applications have all the static and dynamic data stored in relational databases. The aim of this chapter is to generate automatic code for data access located in relational databases in minimum time.

The code generator made by Goodarzi (2009), which is taken the basis of this study was implemented in c# on the .Net platform. It generates SQL code for the database-level part of the code, and c# or VB.Net for the application level code. There are two main approaches to code generation, often referred to as passive and active. The passive approach implies generating code only once (or re-generating it each time a modification is required). The active approach includes the option to automatically update previously generated and manually edited code (Glass 1996; Jacob et al 2006; Jensen et al 2004). The code generator used is the combination of both the approaches.
3.2 PREVIOUS WORK

The database-level code is generated using both approaches. The stored procedures follow the pattern similar to the application-level code; there are automatically generated procedures, which are re-generated each time the generator executes, and there are custom procedures, which are not affected by the generator. However, the tables and their structure are automatically updated. The code generator accepts input a file with the description of the application and processes it in the following steps:

1. A Parser object is responsible for parsing the input and generating a parse tree. The parser is also responsible for validating the syntax and structural integrity of the schema in the input file. The objects constituting the application’s abstract syntax contain detailed validation rules for each part of the application, such as checking that field lengths do not exceed their maximum values, that the data types are database-compatible, etc.
2. A SchemaValidator object is responsible for checking the application schema as a whole, which guards against duplicate class names, duplicate primary keys.
3. A SchemaDatabaseLoader object creates a Database object based on the schema file – which is an abstract model of the database part of the application.
4. An SqlDatabaseLoader connects to the application’s database and does the same based on the schema retrieved from the database. The two abstract databases are compared by a DatabaseComparer object, which insures that the two schemas are compatible (for example, the data type of an existing field cannot be changed to an incompatible data type: a string cannot be converted to an integer, for that might result in loss of data). The DatabaseComparer object exposes several collections,
including tables to create, tables to delete, tables to modify, constraints to create, etc., which are then accessed by objects responsible for generating the actual code.

5. A DatabaseHelper object takes the DatabaseComparer as input, generates all the database-level code, connects to the database and updates it based on the data provided by the DatabaseComparer.

6. An ApplicationLoader object takes the parse tree as input and creates an abstract syntax tree, which is an abstract model of the application. This object is passed on to several objects, which generate the actual code. The data intensive web applications consist of three components:

   1. Data Access Layer
   2. Business Logic Layer
   3. Presentation Layer

The application logic layer is unique for an application and does not contain recurring code patterns. Thus it is not suitable for automatic code generation. Presentation layer is the user interface like desktop application. It does not require automatic code generation.

The data access layer requires automatic code generation for producing recurring data access code that is used by different modules of the web application. Goodarzi (2009) had made a model in XML and code generator made by him generates data access code for Microsoft .NET/SQL server platform. The code generator produces at least 50% of data access code based on specifications provided in the data model but only 20-35% of data access code is used by the application.
3.3 LINK RANK - THE NEW APPROACH

Relational link is a pointer from one data unit to another. There is a lot of code which is unused with respect to application because code generator generates automatic recurrent code used by all applications in a website. Thus to increase the efficiency of data access code, modifications need to be made in data retrieval algorithms. The proposal is to introduce the concept of “Link Rank” between the millions of link and data units in a huge database. It is based on probability distribution.

The data links frequently used by web application are given high ranks than the links between unused data units. For instance, if the database consist of data units A,B,C,D with B and D data links mostly used by the application as shown in Table 3.1.

Table 3.1 Link Rank

<table>
<thead>
<tr>
<th>Links</th>
<th>Number of Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
</tr>
</tbody>
</table>

Since, the probability of access of the data links B and D is very high as compared to other data links A and C. Hence the data links B and D are given high ranks as compared to links A and C. The link rank algorithm is implemented analogous to Google Page Rank algorithm, but it does concentrate only on inbound links and not outbound links. The data links connected by higher ranks relational links are stored in separate servers with the previous servers acting as back-up store. This will allow the code
generator to produce data access code for frequently visited data units. Thus the efficiency of the code generated will increase manifold. This will also lessen the burden of functioning on populated sites.

The java code to rank the links in java platform is given below. This is modified version of the java code generated for Google page ranking by NimaGoodarzi.

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import Jama.Matrix;

public class Ranking {
    private final double DAMPING_FACTOR = 0.85;
    private List params = new ArrayList();
    public static void main(String[] args) {
        Ranking ranking = new Ranking();
        System.out.print(ranking.rank("C"));
    }
}

/* * * solve the equation of ax=b, which: a is the generated matrix based on * * the parameter constants. x is the link ranks matrix. b is a n*1 matrix* * which all the values are equal to the damping factor. */
```
public double rank(String linkId) {
    generateParamList(linkId);
    Matrix a = new Matrix(generateMatrix());
    double[][] arrB = new double[params.size()][1];
    for (int i = 0; i < params.size(); i++) {
        arrB[i][0] = 1 - DAMPING_FACTOR;
    }
    Matrix b = new Matrix(arrB);
    // Solve the equation and get the link ranks
    Matrix x = a.solve(b);
    int ind = 0;
    int cnt = 0;
    for (Iterator it = params.iterator(); it.hasNext();) {
        String curlink = (String) it.next();
        if (curPage.equals(pageId))
            ind = cnt;
        cnt++;
    }
    return x.getArray()[ind][0];
}

private double[][] generateMatrix() {
    double[][] arr = new double[params.size()][params.size()];
    for (int i = 0; i <params.size(); i++) {
        for (int j = 0; j < params.size(); j++) {
            arr[i][j] = getMultiFactor((String) params.get(i), (String) params.get(j));
        }
    }
    return arr;
}

/* This method generates the matrix of the linear equations. The generated matrix is a n*n matrix where n is number of the related pages. */

/* This method returns the constant of the given variable in the linear equation. */
private double getMultiFactor(String sourceId, String linkId) {
    if (sourceId.equals(linkId))
        return 1;
    else {
        String[] inc = getInboundLinks(sourceId);
        for (int i = 0; i < inc.length; i++) {
            if (inc[i].equals(linkId)) { return -1; }
        }
    }
}

/* This method returns list of the related pages. This list is also the parameters in the linear equation*/

private void generateParamList(String pageId) {
    // Add the starting page.
    if (!params.contains(pageId)) params.add(pageId);
    // Get list of the inbound pages
    String[] inc = getInboundLinks(pageId);
    // Add the inbound links to the params list and do same for inbound links
    for (int i = 0; i < inc.length; i++) {
        if (!params.contains(inc[i]))
            generateParamList(inc[i]);
    }
}

/* Return list of the inbound links to a given page.*/

private String[] getInboundLinks(String pageId) {

    // This simulates a simple page collection
    Map map = new HashMap();
    map.put("A", new String[] { "C" });
    map.put("B", new String[] { "A" });
    map.put("C", new String[] { "A", "B" });
    return (String[]) map.get(pageId);
}
3.4 CONCLUSION

The performance of this system needs additional but expensive servers to store the backup of the data which is not retrieved much. The boost in the price of the latest servers / cloud will not be as important when compared to the competence while using the web sites and applications. Moreover, the user interface will also be better.