Agent Generation Programs of Database Systems Automatically

Dr.Yossra Hussain Ali
University of Technology
Bagdad/Iraq
Yossra_h_a@yahoo.com

Dr. Hala Bahjat
University of Technology
Bagdad/Iraq
Hala_bahjat@yahoo.com

Dr.Alia Karim Abdul Hassan
University of Technology
Bagdad/Iraq
hassanalia2000@yahoo.com

Abstract

Software development has become more and more complex, the users expect more features and more flexibility, the projects are getting bigger and bigger, and the software agents are considered as a way for developing software applications. The automatic code generation is not a new concept, and the software development may cause an upsurge in the use of code generators as a way of bringing enterprise software to market extremely quickly.

The idea of this research aims to design a system that generates programs automatically based on software agent technology and is applied in the relational database systems .The reason for choosing this field is the database system technologies have wide applications. All areas of human effort can use and benefit from these technologies.

This project builds a normalized database depending on sample data, where good database design must be matched to good table structures. Table structures are designed and evaluate to control data redundancies, thereby avoiding data anomalies. The process that yields such desirable results is known as normalization. The agent analyzes the sample data and begins the normalization process to design the normalized DB. After this stage the agent builds empty forms (basic structures or templates) that are specific code for DB programs.

The final stage is to fill these empty forms with specific code where it will add new objects to the forms according to the forms function. The addition of these objects is controlled by the agent where it determines its coordinates on the form and writes its specific code. For example the number of textboxes objects is determined by each table's fields. Depending on standard codes which are specific to database operation the agent wrote these pieces of code into the generated forms and made all the necessary changes to them to suite the generated normalized database.

This work uses the Data Access object functions to manipulate objects that are stored in the Database engine.

1. Introduction

An agent is any autonomous system that perceives and acts to achieve a narrow set of goals within specific virtual or real environment [1]. Agent-based approaches appear to offer solutions to the many new challenges faced by information infrastructure. With growth in distributed computing, application programs, and Internet, etc., more and more applications must operate in open systems, where the components change over time, and there is a lot of information available from multiple sources, much of it unstructured.

The use of an agent to build code generation automatically of database systems components leads to configured and extended support of new requirements [6].

Code generation isn't just about writing code quickly. There are many advantages from it [7]:

- Changing the way programmers work: a code tends to be much cleaner and simpler, as it only needs to do what is required now. If requirements change later, simply modify the generator and crank out a new version of the source code.
- Stable and bug-free, "works first time" the debugging takes place when writing the code generator itself–hence the bulk of the debugging work has taken place before the project begins.
- Produced very quickly, just point the generator at the problem domain, and crank out a brand-new, guaranteed up-to-date application program interface (API)
- Customisable (assuming you have the source code for the generator itself, you can easily customise the code that will be generated)
- Takes change in its stride (if a database needs to be changed, simply make the change, then turn the handle and "crank out" a new version of the source code– it takes minutes).
- Programmers are free to concentrate on the areas of development that deserve their brainpower (hence morale improves, which further increases productivity).
- Code mentor, as the generated code is well written and 100% consistent, programmers are more likely to copy the style of this code and learn from it than they would from a dusty old standards document. So all project code is more likely to be consistent and very clear, despite being produced at an incredibly rapid rate.

The user want to detect design or implementation error detection early, i.e. not at run time, but during the generation of the program code and /or subsequent compilation.

2. Theory of Automatic Programming Generation

Computer programming is the process of constructing executable code from fragmentary information that may come in many forms [3]. The result of the programming is a section of code that is capable of receiving inputs from the target domain and processing them to yield appropriate outputs. When computer programming is done by a machine, the process is called automatic generation. Therefore the automatic programming is the generation of programs by computer based on a specification. Figure (1) represents the automatic programming generation [5].

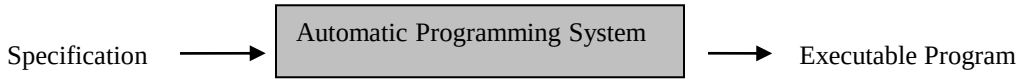


Figure (1): Automatic Programming Generation

There are two approaches to automated software construction known as program analysis and program synthesis. Analysis advocates believe that examining existing programs to see what they do and/or breaking down the programming task into its elementary parts is the best way to arrive at automated programming .Program synthesis is a technique for automatically deriving programs from specifications of their behavior, synthesis proponents argue that it is best to first formally specify a problem by using a very high-level language, their research discovers and articulates the underlying principles of creating or synthesizing software from its specification . A glance through the available literature of Automatic Programming (AP) reveals that program synthesis approach is the more fruitful approach to AP [4].

Most work on program synthesis has focused on deriving applicative programs, which produce an output given an input and have no side effects, program synthesis techniques for imperative programs, which are applied for their side effects, are less well developed than those for applicative programs, research has just begun on techniques for specifying and synthesizing programs that do not terminate, such as operating systems, and programs that are executed on parallel-processing hardware .

The idealized view of AP assumes that the AP systems will have three key features [8]:

- They will be end-users oriented, communicating directly with end users,
- They will be general purpose, working as well in one domain as in another, and
- They will be fully automatic, requiring no human assistance.

Although this view is attractive, it is based on a number of faulty assumptions, so in order not to make AP as a myth and unachievable goal, there are some facts that must be considered:

- End user-oriented AP systems must be domain experts. The system must be oriented toward making reasonable assumptions about unspecified properties.
- Focusing on a narrow enough domains makes it feasible to construct a fully automatic program generator that communicates directly with end users.
- Programming is an iterative process featuring continual dialogue between the end-user and programmer, where the user requirement could not be precise and complete at the first attempt.

3. Program Specification

A generalization of the entire concept of program specification involves an interaction between the user and system as the system builds a model of what the user wants and tries to fill in the details of the algorithm [5]. As shown in the previous section, the goal of program synthesis researches is to develop tools that synthesize efficient programs from program specification. A program specification is the means or methods employed to convey to the AP system the kind of program the user wants. Specifications take a variety of forms, from a verbal description of customer requirements to a set of test cases or an executable prototype. The specification is defined as a precise mathematical abstraction of program behavior. This definition is standard, but our use of specifications is novel [2]. The goal is to make the specification better than programming languages in that it is [2, 3]:

- Smaller
- Easier to write
- Easier to understand (closer to application concepts)
- less error-prone

4. Towards Generative Programming

There are three other programming paradigms, which have goals similar to those of generative programming: Generic programming, Domain-Specific Languages (DSLs), and Aspect-Oriented Programming (AOP). Generative Programming is broader in scope than these, but borrows important ideas from each [8]:

- Generic Programming may be summarized as “reuse through parameterization.” Generic programming allows components, which are extensively customizable, yet retain the efficiency of statically configured code. This technique can eliminate dependencies between types and algorithms that are not necessary.

However, generic programming limits code generation to substituting concrete types for generic type parameters, and welding together pre-existing fragments of code in a fixed pattern. It does not allow generation of completely new code. Generative programming is more general because it provides automatic configuration of generic components from abstract specifications, and a more powerful form of parameterization [8].

- Domain-Specific Languages (DSLs) provide specialized language features that increase the abstraction level for a particular problem domain; they allow users to work closely with domain concepts, at the cost of language generality. Domain-specific languages range from widely used languages for numerical and symbolical computation to less well-known languages for telephone switches and financial calculations.
- Aspect-Oriented Programming (AOP) Most current programming methods and notations concentrate on finding and composing function units, which are usually expressed as objects, modules and procedures.

AOP decomposes problems into functional units and aspects (such as error handling and synchronization). In an AOP system, components and aspects are woven together to obtain a system implementation that contains an intertwined mixture of aspects and components. Weaving can be performed at compile time (e.g. using a compiler or a preprocessor) or at runtime (e.g. using dynamic reflection). In any case, weaving requires some form of metaprogramming. Generative programming has a wider scope that includes automatic configuration and generic programming techniques, and provides new ways of interacting with the compiler and development environment.

5. How Do Code Generators Work [9]

When given the correct conditions, a lot of source code can be automatically generated– the programmer is then free to "fill in the gaps". The question is, under what conditions a code can be automatically generated.

For code to be generated, the following three vital areas must be 100% predictable and well understood: as shown in Figure (2)

1.The Design Patterns

The template to which the code will be produced.

2.The Domain Meta-Data

The topology that we are attempting to model in code, usually augmented with extra data provided by the developer.

3.The Domain Rules

The rules that dictate the structure and behavior of the domain meta-data. This area is normally encapsulated in the generator program itself.

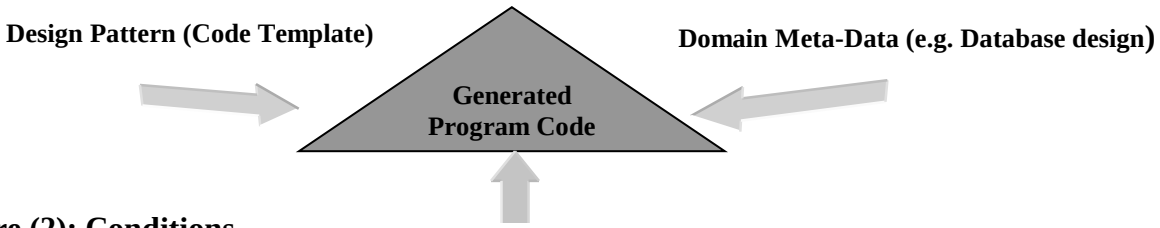


Figure (2): Conditions for Building Code Generation

The Agent Based Automatic Program Generation System is shown in diagram that describe in Figure (3), which consists of many components. Each component is explained in the following sub sections:

1. Generative Engine (Agent)

The Generative Engine represents the important part of this system where the agent deals with sample data to generate the code of database system. This part consists of four parts. The first part represents the sample data that the user supply to the agent, where the agent analysis this data in the second part, then the agent determines the repeated groups and find out all partial and transitive dependencies to create the suitable tables for this data. Part three represents the stage of creating the database for the table that was generated previously to get finally a normalized database. Part four represents the last stage where depending on the database that was created. The agent builds the explicit code for the database system.

2. User

It represents the person who wants to get the programs of the database system. The system can deal with two kinds of users:
(1) Users who do not recognize what is a table or database, therefore the agent helps them by performing the normalization on the sample data entered by them where the agent analyzes the data to determine the fields to build the suitable DB for it. (2) Users who enter tables and fields, in this case the agent directly creates DB system programs.

3. Code Library (Templates)

In this part the templates are stored in the library. These templates represent the empty forms that was created by the agent to distribute the suitable object in the appropriate places and to write the specific code for DBS programs.

4. Code Generation

The agent depends on the project meta-data (database model) and starts to generate the specific code of DBS programs depending on templates where it will make the necessary changes and additions that accommodate with the generated database.

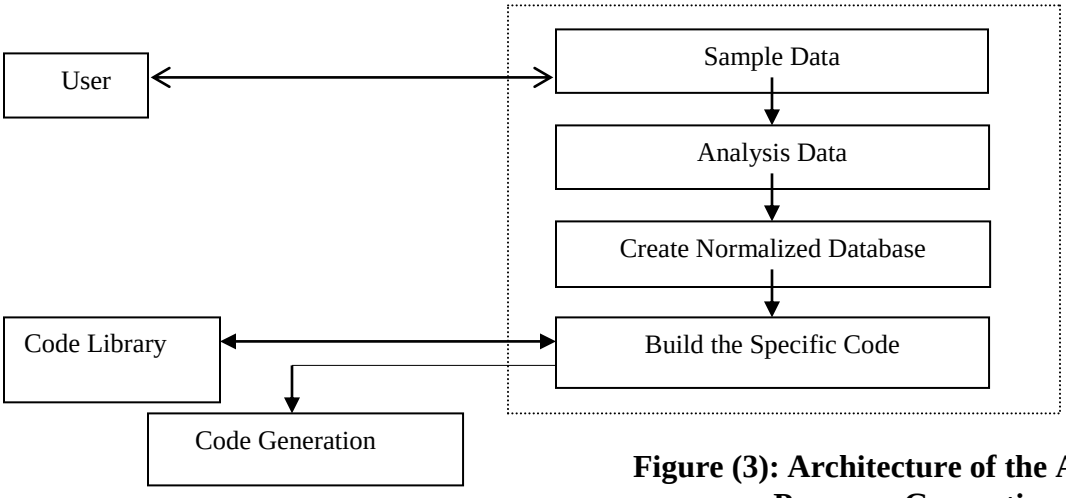


Figure (3): Architecture of the Automatic Program Generation

7. Normalization

The primary goal for this system is to assist and alienate the user from the complexity of dealing with database system, especially if the user does not have any relation with computer science and then does not have any idea about DBS, the basic point in code generation of database system is to generate database. From this, we started to think how to build a normalized database from the sample data provided by the user.

Database normalization can be essentially defined in previous chapter as the practice of optimizing table structures. Optimization is accomplished as a result of a thorough investigation of the various pieces of data that will be stored within the database, in particular concentrating upon how this data is interrelated. An analysis of this data and its corresponding relationships is advantageous because it can result in a substantial improvement in the speed in which the tables are queried.

This section represents how the agent can read sample data from the user and convert it to many tables that are created according to database normalization process. The following example represents how normalization can work through a series of stages called normal form.

Case Table (Sample Data)

Case ID	Case Lname	Control ID	Control Fname	Control Lname	Control Age	Relationship	Case Fame
101	Smith	1001	Fred	Smith	5	Father-son	John
		1002	Larry	Smith	10	Father-son	
		1003	John	Smith,jr.	2	Father-son	
		1004	Margret	Smith	32	Husband-wife	
102	Sanchez	1005	Javier	Sanchez	1	Mother-son	Maria
		1006	Lzel	Sanchez	1	Mother-daughter	
		1007	Juan	Sanchez	44	Wife-husband	
103	Connor	1008	Fred	Connor	25	Wife-husband	Hilary
		1009	Jackie	Conner	2	Mother-daughter	

When looking at the sample data in the case table note that it contains a repeating groups where (Case ID), (Case Fname) and (Case Lname) can have a group of several data entries, which are empty. A relational table must not contain repeating groups. If repeating groups do exist, they must be eliminated by making sure that each row defines a single entity. The process is simple: just add the appropriate entry, in other words, each row and column intersection by one and only one value. Therefore the case table must change to table1.

When one analyzes the data in table1 a search is made to find the primary key. The resulting table (table1) in this example is not relational because it does not have a primary key, to transform this table into relational; a primary key needs to be identified. In this example composite keys (CaseID) and (ControlID) are suitable primary keys for table1 because they have maximum repeated and the type of entities is integer.

Table1 (1NF table)

File Edit Search Help

CaseID	CaseLname	ControlID	ControlFname	ControlLname	ControlAge	Relationship	CaseFname
101	smith	1001	fred	smith	5	father-son	john
101	smith	1002	larry	smith	10	father-son	john
101	smith	1003	john	smith	2	father-son	john
101	smith	1004	margaret	smith,jr.	32	husband-wife	john
102	sanchez	1005	javier	sanchez	1	mother-son	maria
102	sanchez	1006	lzel	sanchez	1	mother-daughter	maria
102	sanchez	1007	juan	sanchez	44	wife-husband	maria
103	connor	1008	fred	connor	25	wife-husband	hilary
103	connor	1009	jackie	connor	2	mother-daughter	hilary
3	3	9	8	4	7	5	3

The numbers under each column in the above table represent the repeation of each entity.

From table1 we must determine the partial dependencies that are based on only a part of a composite primary key, and transitive dependencies that occur when dependency of one non-prime attribute is on another nonprime attribute. Figure (4) illustrates all the dependences that exist in table1.

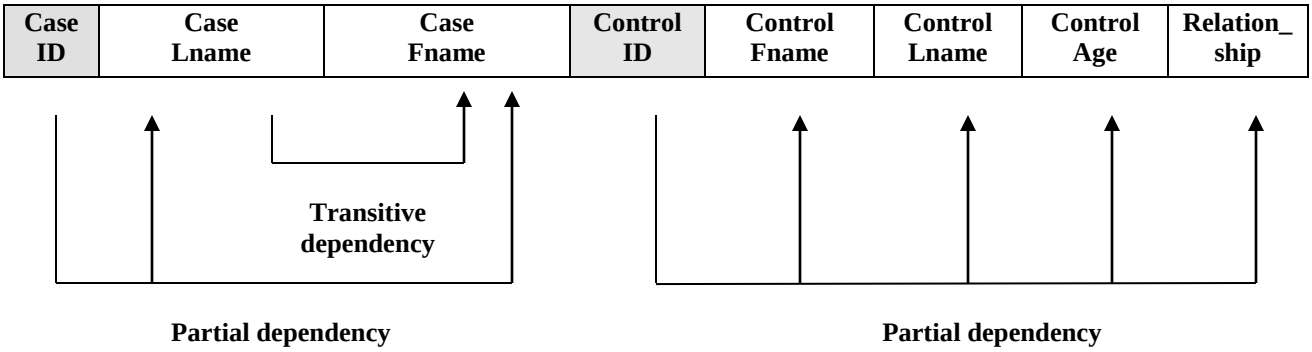
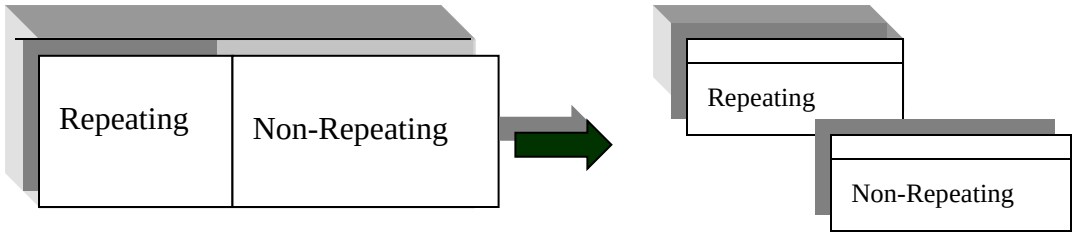


Figure (5): Dependencies for Table1

At the end of this stage the1NF is ended and the first step of 2NF starts.

The process of normalization in general based on the analysis of relations, their schemes, their primary keys and their functional dependencies that describe the relationship between attributes in a relation, therefore the relational table1 must be decomposed or broken down into some other relations that individually meet the criteria of the normal form test; therefore table1 is decomposed into two tables that will replace the original table by:



The 1NF table1 is decomposed in to two tables as shown:

Repeating Table (Table 1)

File Edit Search Help

CaseID	CaseFname	caseLname
101	john	smith
101	john	smith
101	john	smith
101	john	smith
102	maria	sanchez
102	maria	sanchez
102	maria	sanchez
103	hilary	connor
103	hilary	connor

The relationship between the columns in table1 is one to one therefore it can be changed to the following tables:

File Edit Search Help

CaseID	CaseFname	caseLname
101	john	smith
102	maria	sanchez
103	hilary	connor

Non Repeating Table (Table 2)

File Edit Search Help

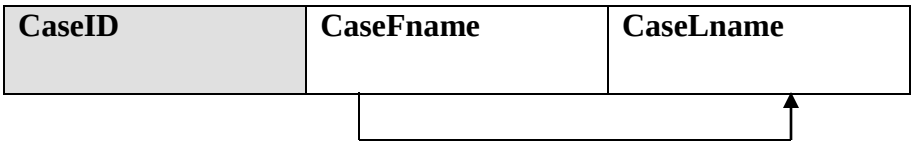
ControlID	ControlFname	ControlLname	ControlAge	Relationship
1001	fred	smith	5	father-son
1002	larry	smith	10	father-son
1003	john	smith,jr.	2	father-son
1004	margaret	smith	32	husband-wife
1005	javier	sanchez	1	mother-son
1006	lzel	sanchez	1	mother-daughter
1007	juan	sanchez	44	wife-husband
1008	fred	connor	25	wife-husband
1009	jackie	connor	2	mother-daughter

These two tables are generated by breaking up any field that contains multiple pieces of information into separate fields and creating a separate table for each set of related data and then identifying a set of related data with a primary key to determine at last the partial dependencies, for that reason the repeated table and non repeated table represent the 2NF of normalization because they do not include partial dependencies.

Now the 3NF must start where it is required that all columns in a relational table be dependent only upon the primary key, in other words, every non-key column in a relational table is non-transitively dependent upon its primary key.

Table2 is already in 3NF,where The non-key columns, (ControlFname, ControlLname , ControlAge and Relationship) are fully dependent upon the primary key (ControID).

Table1 is in 2NF but not in 3NF because it contains a transitive dependency.



Transitive dependency

The process of transforming a table1 into 3NF is identifying any fields that depend on any of the non-key fields of the table, create and name new tables accordingly and assign new primary keys.

Therefore from original table (table1) delete CaseLname and move it to the new table (table3) and CaseFname becomes a primary key to table3.

Table1

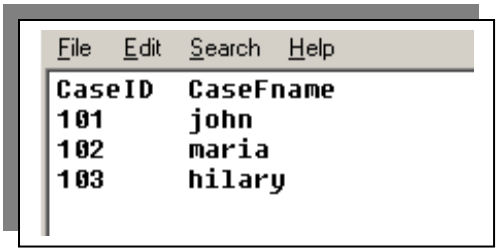
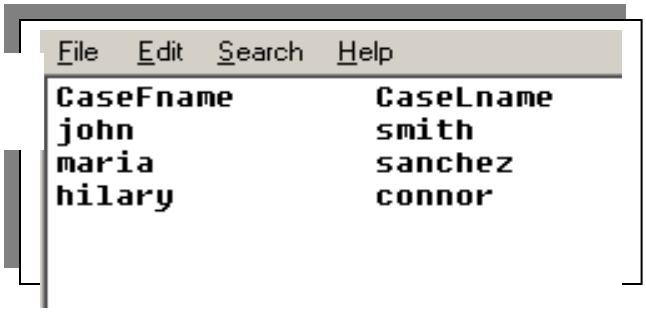


Table3



At the ending of this step three tables from input sample data are obtained. Therefore we create the normalized database for the system that we want to generate code for it automatically.

8. Visual Basic Database

SQL is an industry–standard language for manipulating relational database. SQL tools provides all of the functions that Visual Basic need to write programs that can create, access, query, and modify SQL database.

Visual Basic includes many tools for creating, managing revise, deal with database efficiently. This work deals with Data Access Object (DAO) that represents one of features we find in Visual Basic.

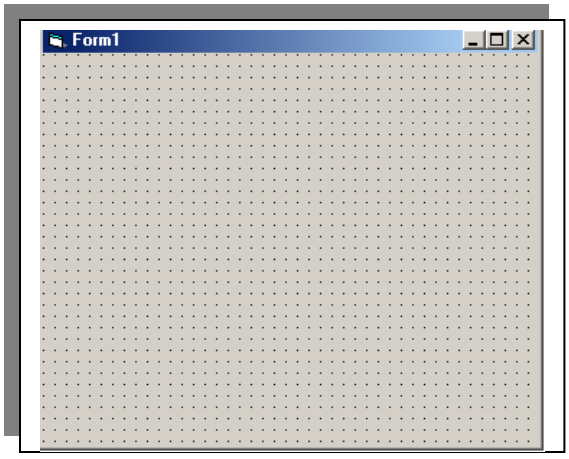
One of main advantages of using DAO is to get to database that DAO serves as a layer of abstraction on top of a specific database implementation. Therefore when a user is programming a database in Visual Basic, he does not write code that talks to the database directly instead he writes code that talks to DAO, which in turn translates the code into something the database understands.

So DAO can be used to manipulate data in Visual Basic code, using DAO, a user can run queries, update values in database tables.

9. Automatic Program Generation

This section clarifies the idea of how automatic program generation system is implemented and how it's agent does the work. Because the generated output code program is in Visual Basic language, it was necessary to study, analyze and understand all contents of objects of this

language. These objects include forms, command bottoms, text boxes, combo boxes, etc.. The study of the contents of objects means how code manipulates these objects. The first of these objects represents the form, after opening an empty form there exists a set of properties and attributes. When an attempt is made to insert other object into this empty form such as a command button we find the same contents of the empty form in addition to the code of the inserted object. Figure (5(a)) represents the empty form, in other words, a form without any inserted objects and Figure (5(b)) shows its code, Figure (6) describes the case when inserting the empty form text box or command button or label button and viewing its code.
When we studied many empty forms we recognized the fact that each empty form takes the same code, and when inserting more than one similar object to the form they will take the same code except the name and location, which are different. From all above, the work in stages of code generation is started.



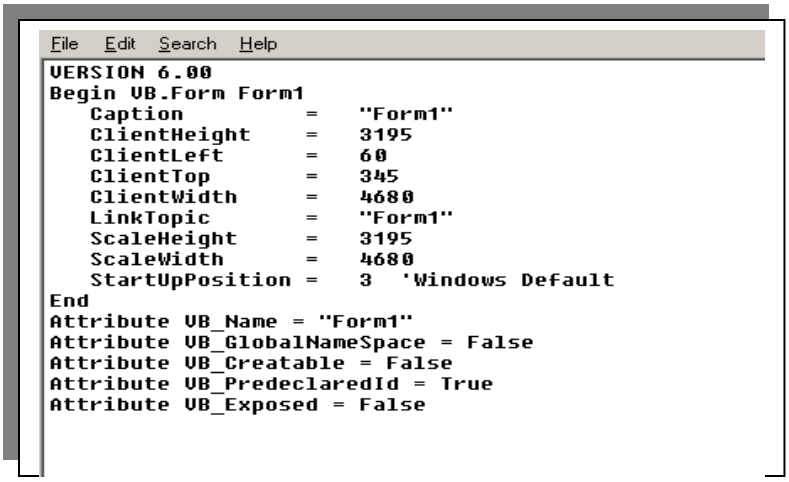


Figure (5(a)): Empty Form

Figure (5(b)): Code for Empty Form

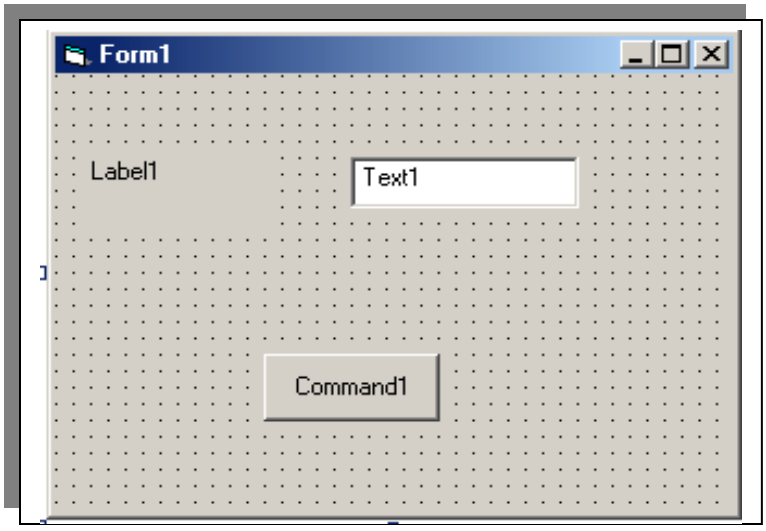


Figure (6(a)): Form with Label, Text and Command

```
VERSION 6.00
Begin VB.Form Form1
   Caption        =   "Form1"
   ClientHeight   =   3195
   ClientLeft     =   60
   ClientTop      =   345
   ClientWidth    =   4680
   LinkTopic      =   "Form1"
   ScaleHeight    =   3195
   ScaleWidth     =   4680
   StartUpPosition =   3  'Windows Default
   Begin VB.TextBox Text1
      Height         =   375
      Left           =   2040
      TabIndex       =   2
      Text           =   "Text1"
      Top            =   600
      Width          =   1575
   End
   Begin VB.CommandButton Command1
      Caption        =   "Command1"
      Height         =   495
      Left           =   1440
      TabIndex       =   0
      Top            =   2040
      Width          =   1215
   End
   Begin VB.Label Label1
      Caption        =   "Label1"
      Height         =   495
      Left           =   240
      TabIndex       =   1
      Top            =   600
      Width          =   1215
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub Form_Load()

End Sub
```

Figure (6(b)): The Code of Form with Label, Text and Command

This empty form represents the template that is used inside the program.
Because there does not exist full code generation programs we need always to create special structure for the job that we want to generate its code, these structures are called templates and their definition is a regular structure that is used to place the generated code inside. To describe this idea, consider the example of an empty paper, that can be filled with any information (application form), this paper plus the setup and its size represents the template. This template is filled with other information automatically according to the type of form (demand information, purchases, sale …etc.) and depending on the coordinates of this paper the fields are distributed and arranged on it.
Depending on this work the agent built the suitable empty forms to start filling them with the information about the database that the user wants.

After building the empty form (templates) for which code will be written, it is necessary to think of building two or more database systems programs like (insert, delete, update, query, display) and study each program alone to determine the similarity aspects among them, for example, the update program must contain the functions for next and previous records in order to move to each record to display the data before updating them. The result of this study is that we

identified the instructions that must exist in any program. This stage represents the first step for the code that is written by the agent inside the empty form. Whereas the aim of this work is to write programs for DBS automatically the agent needs to deal with the DB, which was created in the first part of this system, therefore the agent needs to write all codes for operation with the DB.

After that, the agent starts to build the programs of DBS depending on the results obtained. Therefore the agent must build the interface of each program, which represents the distribution of all objects on the form. Each object has codes and location therefore it must be placed in the suitable coordinates on the form to obtain an acceptable interface. For example we know each program of DBS depends on DB and each DB contains many tables and each table has many fields therefore the agent must allocate the textboxes on the form depending on number of fields. Figure (7) contains three textboxes to describe the location for each textbox, Figure (7(a)) represents the empty form with three textboxes distributed vertically while Figure (7(b)) represents the codes for each textbox. When we study these codes we determine the difference between them. The disparity that we observe between them is in the name and location of each one.

Now starts the idea of how to distribute the objects automatically on the form. To achieve this idea the agent depends on values stored in text files where these values are changed according to simple calculation depending on the type of DB program. For example if we look at Figure (7(b)) the name of textboxes starts with (text0) the agent uses a counter whose initial value is (0) and depending on the number of fields for each table, the agent increases the counter stored in text files. The agent after determining the location of each object, writes the code for the object. This work is illustrated by the algorithms in the next sections.

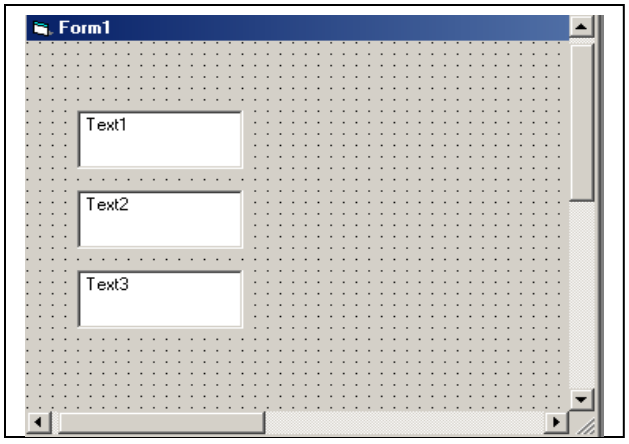


Figure (7(a)): Empty form Container Three Textboxes

```
Begin VB.TextBox Text3
   Height        =  615
   Left          =  480
   TabIndex      =  2
   Text          =  "Text3"
   Top           =  2400
   Width         =  1575
End
Begin VB.TextBox Text2
   Height        =  615
   Left          =  480
   TabIndex      =  1
   Text          =  "Text2"
   Top           =  1560
   Width         =  1575
End
Begin VB.TextBox Text1
   Height        =  615
   Left          =  480
   TabIndex      =  0
   Text          =  "Text1"
   Top           =  720
   Width         =  1575
End
```

Figure (7(b)) Code for Three textboxes

10. Algorithms of the Proposed System

This section implemented to explain the algorithms that describe the system work and the sequence of how the user of a system interacts with it.

The first algorithm presents how normalization is applied to the sample data to get a normalized database that is created by the agent; this agent reads the sample data as input.

Algorithm 1: Normalization

Input: Sample Data.

Output: Normalized database.

Process:

Step1: Agent reads sample data from the user

Step2: While repeating groups exist **Do**

-Set each row/column intersection which contains one and only one value (fill the blank spaces).

End while

Step3: Agent converts sample data to relational table by

-determining the primary key

-identifying each row with a unique or set of primary keys.

Step4: Create separate tables for each group of related data

Step5: Determine the partial and transitive dependencies depending on relationship between the entities.

Step6: Remove subsets of data that apply to multiple rows of a table and place them in separate tables (partial dependency).

Step7: Create relationships between these new tables.

Step8: Remove columns that are not dependent upon the primary key (transitive dependency).

Step9: End Algorithm.

The second algorithm presents the idea of how to design patterns of empty form that represents the templates that are stored in code library.

Algorithm 2: Generate Forms

Input: Number of forms that are required to build the DBS (N).

Output: Build templates for empty forms.

Process:

Step1: Open text file for writing.

-Print inside file:

-Properties that are specific for empty form.

-Attributes that are specific for empty form.

Step2: End Algorithm

The previous algorithm clarifies how to generate an empty form. The properties and attributes represent the features of each form that will be acceptable for compilation.

The next algorithm (interacting with the system) represents the second type of user who does not depend on the agent to convert the sample data to the relational tables.

Algorithm 3: Interacting with System

Input: Database name, tables, fields, and type of keys.

Output: Build total normalization database.

Process:

Step1: Agent demands the following input to do the work.

- Choose the place to save the database depending on drive, directory, and file.
- Determine name of database.

-If name of database exists **Then**

-Print warning to change the name this database.

End if

Step2: Creation Database.

Repeat

Create Table by:

For I = 1 to number of fields **Do**

-Input field name, type, size, and type of key

-Create Field (field name, field type, field size).

- If field represents an equation **Then**

- Open text file to write this equation.

End if

Next I

Until no tables

Step3: End Algorithm

In the end of previous algorithm, the interaction with the user is terminated therefore the other algorithms represent the structures of programs that are generated by the agent in this system.

The following algorithm presents the creation of the main form in the system as follows:

Algorithm 4: Generate Interface DBS Form

Input: Number of programs that are specific for DBS (N), text files which contain properties and attributes of Commands Buttons, Textboxes and Combo Boxes.

Output: Generate the main form

Process:

Step1: Do (generate form algorithm) to build the main form (FF1).

Step2: Determine Initial Value for Height, Left, TabIndex, and Width.

Step3: Open (FF1) for Append.

While not EOF (FF1) **Do**

-Search for word "Caption"

If found Caption **Then**

- Print (FF1): Caption = "Main Form"

End if

End while

For I= 1 to N **Do**

-Open (FF1)

-**Open** (F2){commands button text file from code library}
While not EOF (F2) **Do**
-Search for word "StartUpPosition" in (FF1)
-**If** found **Then**
- Read (F2): textline
- Print (FF1): textline
End If
If I= 1 **Then** Print (FF1) Caption "*insert"*
End if
If I=2 **Then** Print (FF1) Caption ="*delete"*
End if
If I=3 **Then** Print (FF1) Caption ="*update"*
End if
If I=4 **Then** Print (FF1) Caption =" *query"*
End if
If I=5 **Then** Print (FF1) Caption =" *display"*
End if
-Change the Initial Value for Height, Left, TabIndex, and Width Depending on the previous stored values that are computed to determine the Location.
End While
Next I
-K=1
Repeat
- Search for word "End"
If found "End" **Then**
- Print (FF1): "Privet Sub Command (K),"_Click ()"
- Print (FF1): "Form (K). Show"
- Print (FF1):" End Sub"
End If
Until no word "End"
Step 4: End Algorithm

The following algorithm presents the creation of the insert part in the database system.

Algorithm 5: Insert Program
Input: Normalized DB, text files which contain the properties and attributes of commands button, textboxes, and combo boxes (F2) and text file containing the standard code for insert program (X1).
Output: Generated code for insert program.
Process:
Step1: Do (generate form algorithm) to build insert form (FF2).
Step2: Determine Initial Value for Height, Left, TabIndex, and Width for Textboxes and Commands.
Step3: Open (FF2) for Append.
While not EOF (FF2) **Do**

-Search for word “Caption”
If found Caption **Then**
- Print (FF2): Caption =”I*nsert”*
End if
-**Open** (F2) {commands button text file from code library}
While not EOF (F2) **Do**
- Search for word “StartUpPosition” in (FF2)
If Found **Then**
- Read (F2): textline
- Print (FF2): textline
End if
-Change the Initial Value for Height, Left, TabIndex, and Width Depending on the previous stored values that are computed to determine the Location.
End while
-**Open** (F3) {combo box text file from code library}
While not EOF (F3) **Do**
- Read (F3): textline
- Print (FF2): textline
-Change the Initial Value for Height, Left, TabIndex, and Width Depending on the previous stored values that are computed to determine the Location.
End while
-Depending on the user selection of tables name
For I = 1 to Number of Fields **Do**
-**Open** (F4) {textboxes text file from code library}
While not EOF (F4) **Do**
- Read (F4): textline
- Print (FF2): textline
-Change the Initial Value for Height, Left, TabIndex, and Width Depending on the previous stored values that are computed to determine the Location.
End while
Next I
While not EOF (X1) **Do**
- Read (X1): textline
- Print (FF2): textline with necessary changes
End while
End while

Step4: **End** Algorithm

The other algorithms work in same manner to achievement the delete, update, and query and display program for database system.

12. Conclusion

From this research and the system design and implementation phases, some conclusions are obtained, they are listed below:

1. The software agent generates normalized database that is considered as the main step in any DB design.
2. The code generated by agent in this work is always correct, because it depends on the templates that are stored in the library and also on the derivation of formula from a more fundamental statement of the problem.
3. Programming time is saved, because a lot of repetitive code can be derived completely from a more compact code and as bugs or shortcomings are found they can be fixed in code templates and applied across the code base.
4. The agent offers the possibility to use fields in the DB that take their values from equations of other fields in the same table.
5. The productivity gains of code generation lie in the agility of the code generators to rebuild the code base, and the requirements of the project do not change. One generation passed can add or remove large segments of code.

13. References

[1] Bradshaw J., "**An Introduction of Software Agents** ". http://www.registratiekamer.nl/bis/top_1_5_21.html. / 1998.

[2] Browne C.J. , "**Automatic programming**". http://www. CS.utexas.edu / 2004.

[3] Castanares Anthony," **Program Generation**". http://www.cs.rice.edu/~taha/teching / 14 Jan 2005

[4] Galmiche D., "**Constructive System for Automatic Program Synthesis**", Theoretical Computer Science, Vol. 71, No. 2, 1990.

[5] Hassan F. R., **Automatic Syntheses of Simple Prolog Programs**, M.Sc. Thesis, Department of Computer Sciences, University of Technology, September 1994.

[6] Hyacinth S. Nwana , "**Software Agent: An Overview**", Knowledge Engineering Review, Vol. 11, No. 3 PP. 1-40,sept 1996.

[7] Jack D. Herrington, "**Code Generation: The One Page Guide**", Code Generation Network (CGN). http://www code generation.net/files/javaone.pdf / 2003.

[8] Krzysztof Czarnecki, Ulrich Eisenecker, Robert Gluck, David Vandevoorde, Todd Veldhuizen, "**Generative Programming And Active Libraries**", Department of Computer Science, Indiana University. http://osi.iu.edu/~treldhui/papers/dagstuh/1998/dagstuhl.html / 1998.

[9] Stephens Matt, "**Automated Code Generation**", software Reality. http://www. Softwarereality.com/programming/code. / May 6, 2002.