J. King Saud Univ., Vol. 21, *Comp. & Info. Sci.*, pp. 1-10, Riyadh (2009/1430H.)

Formalization and Verification of Relational Database Normal Forms Using the Gamma Framework

Hassan Mathkour
Department of Computer Science,
College of Computer & Information Sciences,
King Saud University, Riyadh, Saudi Arabia
mathkour@ccis.ksu.edu.sa, binmathkour@yahoo.com

(Received 21/12/2008; accepted for publication 02/02/2009)

Keywords: Gamma, Parallel processing, Relational databases, Normalization, Data normal forms, Top-down approach, Bottom-up approach, Gamma Virtual Machine.

Abstract. The General Abstract Model for Multi-set mAnipulation (GAMMA) is a parallel computational and programming paradigm. It utilizes the multi-set data structure and a program structure that is defined as a pair of <condition, action>. The elements of the multi-sets are consumed in successive chemical reactions to produce new elements according to a set of conditions. In this paper, we exploit the expressiveness of Gamma to elegantly and succinctly specify the normalization aspects in relational databases and use its computational power to achieve greater performance in verifying and realizing relational database normal forms. We present two approaches in database designs and discuss the performance of Gamma on the extensive computation involved in the relational database normalization process.

1. Introduction

The production of high performance software has become a necessary and important aspect in reducing the rising cost of computing arising from the limitations of sequential execution of most software systems. One approach to combat this rising cost is to apply parallelism into the production (design and implementation) of software to ensure its efficient computation at minimal cost.

GAMMA (Banâtre *et al.*, 2000; Berry and Boudol, 1992) has been found to be an efficient and simple tool with high-level abstraction best suited to achieve high level of parallelism in the production of computer applications. An interesting feature of GAMMA is its very powerful capability to express parallel specifications in a simple manner, devoid of details required for the implementations of such specifications.

A wide range of today's applications incorporates databases upon which they build their functionalities. Hence the high performance and the integrity of a database are indeed very crucial to any of these applications (Elmasri And Navathe, 2007; Wong *et al.*, 1998). Database normalization is a procedural technique of ensuring the elimination of inconsistencies/redundancies that could affect the database integrity (Diederich and Milton, 1998; Elmasri and Navathe, 2007).

In this paper, we present not only our effort in achieving high performance databases by applying the GAMMA framework to the normalization process in the database design, but also the different technicalities involved in applying this framework to these normalization processes. In addition, we give from the Gamma point of view the specification of relational database normal forms that separates the concerns of architecture and implementation issues from the task of developing a correct solution for the normalization problem. Specifically, we (i) present two approaches to verify relational database normal forms including Boyce-Codd normal form (BCNF) (Codd, 1970; Codd, 1972; Codd, 1974) using Gamma, (ii) show how the Gamma specification mechanism is employed to succinctly decompose a set of attributes, taking into consideration a set of functional dependencies, into relational database normal forms, (iii) empirically compare the

performance of the two approaches using a fairly large number of attributes representing real data, and (iv) present experimental results to show the performance gained using the Gamma parallel processing capability compared to that of the sequential processing of relational database normalization.

The first approach of the two normalization approaches presented here is a top-down approach where a set of attributes are verified against the conditions of Boyce-Codd normal form (BCNF) with respect to a set of functional dependencies. If the BCNF cannot be attained, a verification of the third normal form (3NF) takes places. If the 3NF fails, decomposition into the 3NF that is lossless and which preserves the set of functional dependencies is obtained. The second approach is a bottom-up approach where the first normal form (1NF), the second normal form (2NF), and the third normal form are verified respectively. We observe that both approaches made it succinctly possible using Gamma. We experiment with the specifications on a virtual machine implementing the Gamma paradigm to demonstrate the computational performance gained from the natural parallelism of the Gamma paradigm. We show a superior performance as compared to the sequential machine. We also highlight a comparison between the top-down approach and the bottom-up approach in our experimental work.

The remaining of the paper is organized as follows: Section 2 presents related work and briefly describes Gamma. Section 3 presents the Gamma-based top-down approach. Section 4 presents the Gamma-based bottom-up approach. Section 5 highlights the implementation of a Gamma virtual machine on top of a Java machine. Section 6 reports on the performance of the Gamma machine on processing the normalization aspects of the relation database. Section 7 concludes the paper.

2. Related Work

Since the introduction of the relational database model (Codd, 1970; Codd, 1972; Codd, 1974), the automation of the normalization was the subject of several papers (Ling and Goh, 1992; Maier, 1983; Wong *et al.*, 1998). Most researches have shown that to achieve good design, relations must be normalized into at least third normal form (3NF) (Codd, 1974). Several researchers devoted their efforts on normalization process on the design of databases, some of which were reported to have resulted into slow computation (Diederich and Milton, 1988), while some other efforts (Akehurst, 2002; Connolly and Begg, 2002) have resulted into improved performance. Ling and Goh (1992) improved on database normalization rules in order to enhance database design improvement. The rules entail normalizing relations into 3NF, which is then subjected to a further normalization called the Inclusion Normal Form (IN-NF). Wong *et al.* (1998), however, suggested that correct normalization of a database schema must be associated with functional dependencies to determine the attribute semantics that were mined by the database designer. This procedure may be difficult to follow especially when the designer is faced with a large number of attributes for which no semantic information is provided. They (Wong *et al.*, 1998) propose an algorithm for mining dependencies in observed data. The work in Omiecinski (1990) is an effort to improve the performance of automated normalization by applying parallel algorithms for computing the minimal covers and synthesizing relations into 3NF. Short of performing the decomposition into normal forms, Touir *et al.* (2009) present a bottom-up approach using Gamma to verify the conformance of a given relational database design to normal forms up to the third normal form. They also present an experimental evidence of the superiority in performance using a Gamma virtual machine. In their experimental study, they (Touir *et al.*, 2008) use a modest machine specification compared to current available machines.

The most interesting feature of Gamma is its formalism which places no restrictions on how data elements are to be manipulated. Generally, the Gamma paradigm (Berry and Boudol, 1992, Wong *et al.*, 1998) is based on the concept of multi-sets, which in practical terms, behaves like a chemical solution, where its elements are seen as molecules. A Gamma program, which consists of a pair of *Reaction Condition* and *Action*, executes by continually and progressively replacing the multi-set elements that satisfy reaction condition by the products of the action, until a stable state is reached, when no more reactions can take place. Thus, execution of a GAMMA program, which is basically interactions between elements of its e multi-set are non-deterministic, resulting in programs that are capable of executing naturally and implicitly in parallel. An example of a Gamma program is specified as follows:

$$x_1, x_2, \ldots, x_n \rightarrow \phi(x_1, x_2, \ldots, x_n) \Leftarrow \Psi(x_1, x_2, \ldots, x_n)$$

where $x_1, x_2, \ldots, x_n$ is the set of elements (multi-set) that cause the "chemical reaction", Ψ is the reaction condition that must hold within the solution, and ϕ is the action to be taken when Ψ is true. Gamma has

been applied to various domains including string processing problems, graph problems, mathematical problems, geometric problems, image processing applications, reactive programming and software architectures (Berry and Boudol, 1992; Inverardi and Wolf, 1995; Wong *et al.*, 1998).

As a simple example to illustrate how a Gamma program is defined, consider the problem of computing the sum of *n* elements. In this case, we specify the program that computes the sum of two elements of the set, and replace them by the resulting sum. The program can be defined as follows:

Program name: *sum*
Elements causing the chemical reaction: x, y
The reaction condition: $z = x + y$
The action: z
Therefore, the program would be written as:
sum: $x, y \rightarrow z \Leftarrow (x + y)$.

3. A Gamma Top-Down Normalization Approach

Given a set of attributes R = (x1, …, xn) and a set of functional dependencies F = {fd_1, …, fd_m}, R is verified against F for the highest, desired normal form. The highest desired normal form in our treatment of this approach is the BCNF (Codd, 1974; Connolly and Begg, 2002; Elmasri and Navathe, 2007). R is considered to be in BCNF if every determinant is a candidate key. If the BCNF is not attained, the third normal form (3NF) is accepted. The 3NF relaxes the conditions of the BCNF but insists that every attribute in R is dependent only upon the primary key. The procedure that is followed in our top down approach is given in the following steps:

1. Take R and F as input.
2. If R satisfies the conditions of the BCNF with respect to F, then stop and exit; otherwise proceed to Step 3.
3. Verify that R against the conditions of the 3NF with respect to F. If R satisfies the conditions of the 3NF with respect to F, then stop and exit; otherwise proceed to Step 4.
4. Decompose R to obtain a 3NF design that is lossless and which preserves the functional dependencies.

The above procedure is succinctly specified as a GAMMA program as follows (referred to hereto after as the top down program). The notations adopted are given in Table 1. The procedure assumes that the attributes of R conforms to the atomicity property; i.e., R is in the first normal form (1NF). The Gamma program to verify and construct the first normal form in case the attributes are not atomic are given below.

Table 1. Adopted notations

$x_1, x_2, \ldots, x_n$	A set of elements (multiset) which cause the "chemical reaction"
ϕ	Action
Ψ	Reaction Condition
R	A relation to be processed
F	A set of functional dependencies
fd	Defined below

Definition: For the purpose of our specification, a functional dependency *fd* is envisaged in the following notation:

fd(InitialRelation, CreatedRelation, LHS, RHS, Keys, 1NF, 2NF, 3NF, BCNF), where InitialRelation is the relation that has *fd* as functional dependency.

CreatedRelation is initially a null relation that may be created if needed.

LHS is the set that contains the name of attributes in the determinant of the functional dependency.

RHS is the set of attributes that are dependent on the LHS determinant in the functional dependency.

Keys is the set of defined candidate keys in the relation.

1NF, 2NF, 3NF, BCNF are Boolean variable that indicate whether the relation is in 1NF, 2NF, 3NF, BCNF or not. These fields are initially set to false.

Gamma-TopDown-Approach(($\Psi 1, \phi 1$), ($\Psi 2, \phi 2$), ($\Psi 3, \phi 3$)(*fdi*)
Elements causing the reaction: fd_i
The reaction condition Ψ *1: BCNFCheck*((fd_i) and (fd_i.*BCNF = True*)
The action ϕ 1: {$fd_1, fd_2, fd_3, \ldots, fd_n$} – {$fd_i$} // in fact it is removed automatically
The reaction condition Ψ 2: (fd_i.*BCNF = False*) and *3NFCheck*(fd_i) and (fd_i.*3NF = True*)
The action ϕ 2: {$fd_1, fd_2, fd_3, \ldots, fd_n$} – {$fd_i$}
The reaction condition Ψ 3: *3NFDecompose* (fd_i)) and (fd_i.*3NF = False*)
The action ϕ 3:.*modify*(fd_i) // the complete specification is shown below
Therefore, the Gamma program is:
(**) *GammaGeneralCheck*: ($fd_i \rightarrow$ {$fd_1, fd_2, fd_3, \ldots, fd_n$} – {$fd_i$} $\Leftarrow$ *3NFDecompose (3NFCheck (BCNFCheck(fdi) and* (fd_i.*3NF = True)))*

The only statement needed for the actual Gamma program is the statement with (**). The statements preceding it are given for the purpose of clarification. We follow the same style throughout the paper. It is observed that the expressiveness of GAMMA made it possible to specify the complete top down normalization process in an elegant and succinct fashion. The top down program is explained below.

The Gamma program consists of sub-programs that echo the corresponding verification steps as detailed below. The first step is to ascertain that all attributes are atomic; i.e., R is in 1NF. This is necessary for the procedure to obtain the highest normal form from the top down.

3.1. 1NF verification program

According to the definition of first normal form (1NF), we have to ensure that every attribute A in R holds atomic values. For this purpose, an attribute is defined as the pair (attribute name, atomic), where atomic is a Boolean valued variable to indicate the atomicity. Hence, the predicate *AtomicAttribute* is defined as follows:

AtomicAttribute: checks that an attribute in R holds atomic values.

The Gamma specification would be as follows:

Program name: 1NFCheck

Elements causing the chemical reaction: attribute A

The reaction condition R_check: *AtomicAttribute (A)*

The action A_check: *A.atomic =True*

Therefore the Gamma program is:

1NFCheck: A.atomic = True $\Leftarrow$ *AtomicAttribute(A)*

In case the attributes do not conform to the atomicity property, decomposition takes place to break them down into atomic values as follows.

3.2. 1NF decomposition

The 1NF decomposition process has a reaction that triggers the action if the field *atomic* is false and does nothing if it is true. It comprises breaking down each composite attribute A of R into atomic ones.

The specification program would be:

Program name: *1NFDecompose*

Elements causing the reaction: *attribute* A_i

The reaction condition: $A.constituents = break\_down(A_i)$,

The action: *A.constituents*

The corresponding Gamma program would be:

1NFDecompose: $A_i \rightarrow A_i.constituents \Leftarrow break\_down(A_i)$ *and replace A by A.constituents in* fd_i

3.3. BCNF verification program

The Gamma program below verifies that the left hand side of each functional dependency is a key; otherwise the BCNF stays false.

The Gamma specification would be as follows:

Program name: *BCNFCheck*

Elements causing the reaction: fd_i

The reaction condition: $(\lambda \subset fd_i.LHS \mid \lambda \in fd_i.Keys)$

The action: $fd_i.BCNF = True$

The corresponding Gamma program would be:

BCNFCheck: $fd_i \rightarrow fd_i.(BCNF = True$ *and* $\{fd_1, fd_2, fd_3, \ldots, fd_n\} - \{fd_i\}) \Leftarrow (((\lambda \subset fd_i.LHS) \mid \lambda \in fd_i.Keys))$

3.4. 3NF verification program

To verify the 3NF, the Gamma program checks that for each functional dependency the following holds:

- Either the left hand side contains a candidate key
- Or the right hand side is part of a candidate key.

More succinctly:

- $\lambda \subset fd.LHS \mid \lambda \in fd_i.Keys$
- $fd.RHS \in \lambda \mid \lambda \in fd_i.Keys$

The Gamma specification would be:

Program name: *3NFCheck*

Elements causing the reaction: fd_i

The reaction condition: $(\lambda \subset fd_i.LHS \mid \lambda \in fd_i.Keys)$ *or* $(fd_i.RHS \in \lambda \mid \lambda \in fd_i.Keys)$

The action: $fd_i.3NF = True$

Therefore the corresponding Gamma program would be:

$3NFCheck: fd_i \rightarrow fd_i.(3NF = True$ *and* $\{fd1, fd2, fd3, \ldots, fdn\} - \{fdi\}) \Leftarrow (fdi.BCNF =$ false) and $(fd_i.1NF = True)\ \})$ *and* $((\lambda \subset fd_i.LHS)$ *or* $(fd_i.RHS \in \lambda) \mid \lambda \in fd_i.Keys))$

In case the given set of attributes fails to meet the BCNF or the 3NF conditions, it is decomposed into smaller sets of attributes representing a set of new relations that satisfy the 3NF conditions, are lossless, and the set of functional dependencies are preserved. Such decomposition is performed as follows.

3.5. Specification of the 3NF decomposition process

The decomposition process has a reaction that triggers the action if the Boolean variable 3NF is false and does nothing if it is true. The principle of the decomposition comprises creating a relation for each fd where the key is the left hand side of the dependency.

The Gamma specification would be:

Program name: *3NFDecompose*

Elements causing the reaction: fd_i

The reaction condition: $(fd_i.3NF = False)$

The action:

$fdi.Relation = new\ Relation(fdi.LHS \cup fdi.RHS)$, (i)

$fdi.is\_Key\_in = fdi.keys \subset (fd_i.LHS \cup fdi.RHS)$ (ii)

$fdi.keys = fdi.LHS,$ (iii)

$fd_i.3NF = true$ (iv)

$\{fd_1, fd_2, fd_3, \ldots, fd_n\} - \{fd_i\}$ (v)

Referring to the reaction condition and the actions (i-v) in the numbered statements above, the reaction checks if the *fdi* is in 3NF and if not (i) a new relation is created. This latter is composed of the union of the left hand side and the right hand side of the dependency; (ii) checks whether the key of the *fdi* is in the newly created relation, (iii) the left hand side of the *fdi* becomes the key of this new relation; (iv) sets the Boolean variable 3NF to true, (v) *fdi* is removed from the list of original functional dependencies.

4. A Gamma Bottom-Up Normalization Approach

Atomic(A)	True if A is atomic
Name(A)	Refers to the name of the attribute A
τ(A)	Refers to the type of the attribute A, which is either Prime or NonPrime
Δ(A)	Is a set of attributes on which the attribute A is dependent
Ω(A)	Is the set of all attributes constituting the primary key (including A) if A is prime; Empty set if A is nonprime

Using the above definition, we define the 3NF as follows:

For any three attributes (A, B, C), if $\tau(A) = \text{Prime} \wedge \tau(C) = \text{nonprime} \rightarrow (A \in \Delta(C) \wedge B \in \Delta(C))$ does not hold. As illustrated in Fig. 1, having a set of attributes $A_1, A_2, A_3, \ldots, A_n$, we first check if every A_i is atomic; if not, make it atomic using 1NFDecomposition program defined in Section 3. Then, we agglomerate the obtained set of attributes into a set of all possible triplets. Following that, we check if every triplet satisfies the 2NF condition; if so we proceed to the 3NF. This is summarized in the following steps.

1. Start with the 1NFCheck program, namely: $1NFCheck: A_i \rightarrow atomic = False \Leftarrow \neg\, AtomicAttribute(A_i))$
2. If required, use the 1NF Decomposition program (given in Section 3) to decompose each non-atomic attribute A into a set of attributes.
3. Agglomeration steps: We defined a process *CreateTriplet* that creates a multi-set of all possible triplets from the set of 1NF valid attributes A_i, i = 1, …, n. The specification of such a process is as follows:
   Program name: *CreateTriplet*
   Element causing the reaction: A_i, A_j, A_k
   The reaction condition: $triplet_i = new\ Triplet(A_i, A_j, A_k)$
   The action: $triplet_i$

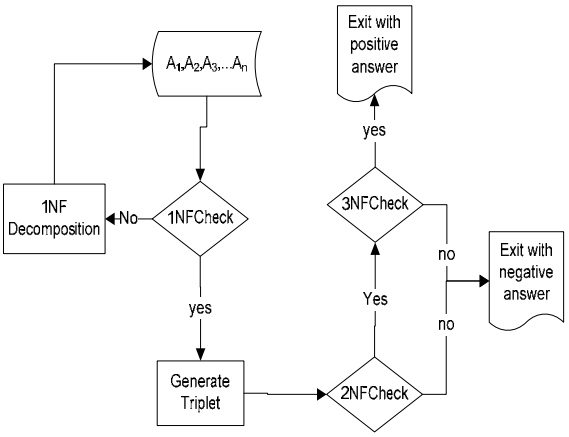

Fig. 1. Steps of the design process in the bottom-up approach.

The next step is ascertaining that every relation conforms to the second normal form (2NF). That is, every NonPrime attribute is fully functionally dependents on the primary key (no NonPrime attribute is partially dependent on the primary key). Therefore, the conditions for Gamma routine for the 2NF are:

- *Having a triplet (A, B, C)*
- *Triplet.τ(A) =Prime and triplet.τ(B) = Prime and triplet.τ(C) = NonPrime*
- *triplet.A.Ω and triplet.B.Ω are equivalent*
- *triplet.A and triplet.B ∈ triplet.Δ(C)*

The action is: $triplet_i$.2NF = True

The complete Gamma program for the 2NF would be:

Program name: *2NFCheck*
Element causing the reaction: $triplet_i$
The reaction condition: $triplet_i.2NF = true$
The action: $(triplet_i.\tau(x) = P) \wedge (triplet_i.\tau(y) = P) \wedge (triplet_i.\tau(z) = N) \wedge (triplet_i.\Omega(x) = triplet_i.\Omega(y)) \wedge (triplet_i.x \in triplet_i.\Delta(z)) \wedge (triplet_i.y \in triplet_i.\Delta(z)) \wedge (triplet_i.\tau(z) = N) \wedge (triplet_i.\Omega(x) \equiv triplet_i.\Omega(y)) \wedge (triplet_i.x \in triplet_i.\Delta(z)) \wedge (triplet_i.y \in triplet_i.\Delta(z))$

Compared to the specification given in (Touir *et al.*, 2008), this specification is more natural and succinct. Similarly, for the 3NF we need to verify that:

- The relation is in the second normal form $triplet_i$.2NF = True
- No NonPrime attribute is transitively dependent on the primary key. That is:
  - $triplet_i.\tau(x) = Prime$ and $triplet_i.\tau(z) = NonPrime$
  - $\neg (triplet_i.x \in triplet_i.\Delta(y)$ and $triplet_i.y \in triplet_i.\Delta(z))$

The action is: $triplet_i$.3NF = True.

Therefore, the Gamma program for the 3NF would be:

Program name: *3NFCheck*
Element causing the reaction: $triplet_i$
The reaction condition: $triplet_i.2NF = true \wedge triplet_i.3NF = false$
The action: $(triplet_i.2NF = True) \wedge (triplet_i.\tau(x) = Prime) \wedge (triplet_i.\tau(z) = NonPrime) \wedge (triplet_i.x \in triplet_i.\Delta(y)) \wedge (triplet_i.y \in triplet_i.\Delta(z))$

The complete Gamma program of the bottom-up normalization approach is defined in the following manner.

1NFDecompose: $A_1, A_2, A_3, \ldots, A_n \rightarrow \{A_1, A_2, A_3, \ldots, A_n\}-\{A_i\} \Leftarrow 1NFDecomposition(1NFCheck(A_i))$

Gamma_Norm: $A_1, A_2, A_3, \ldots, A_n \rightarrow \{triplet_1, triplet_2, triplet_3, \ldots, triplet_n\} - \{triplet_i\} \Leftarrow 3NFCheck(2NFCheck((CreateTriplet(A_1, A_2, A_3, \ldots, A_n)))) \wedge triplet_i.3NF = True.$

5. The Environment of the Experimental Studies

The experiment is conducted using a Gamma Virtual Machine (GVM) which was developed on top of a Java Virtual Machine (JVM). The Gamma concept was implemented as a Gamma Virtual Machine (GVM) to study the efficiency and behavior of a non-Von Neumann machine. The GVM was utilized to express and experiment with the specifications of the database normalization process discussed earlier in order to measure their performance. Since Gamma is based on the concept of multi-set, the GVM is a set of workers that interact simultaneously to process the data on the multi-set to solve a specific problem. The architecture of GVM is shown in Fig. 2. It was developed with the following assumptions in mind:

- The GVM is composed of a set of workers that run on top of a Java Virtual Machine (JVM).
- Each worker is launched on a thread.

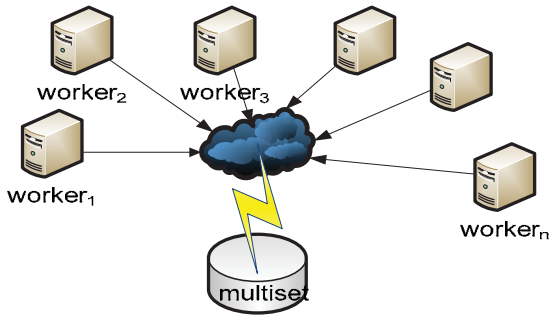

Fig. 2. Interaction principle between the different workers.

The design of the GVM is dependent on the use of workers. Workers access the multi-set concurrently to achieve the parallel processing requirement of the Gamma specification.

The basic components that must be created first are those that are associated with the multi-set elements and those that are implementing the reaction-action pair as illustrated in Fig. 3 and explained below:

- This *FDMolecule* class represents a *molecule* (element) in the multi-set. The molecule deals with a single functional dependency in the relation.
- The *FDProcess* class includes the two required methods, *getMoleculeCardinality()* and *action()*. It is a main component. It provides the necessary interfaces and communication between the molecule and the GVM. The GVM needs a method called *action* as an entry point to start the process. Figure 4 illustrates a sample code implementing the *action* of the gamma program *3NFDecompose* given earlier.
- The *FDManager* class serves as a medium between the user and the GVM. It accepts the list of functional dependencies of the relation in question along with the set of keys as input. It initializes the multi-set, so that it is ready to be invoked by the different workers of the GVM.
- GammaFD is used to instantiate the machine with the desired number of workers. It initiates the process and invokes the machine with a set of functional dependencies of a relation along with the set of keys as an input in order to obtain the results.

In the method action, there are three linked lists: atoms, activeAtoms, passiveAtoms. These three structures play the role of multi-lists of the FDs. Each time an FD is extracted from the atoms list, it is processed and added into the passiveAtoms list. Although the activelist should be present in the implementation of the Gamma machine independently of any problem, it is not used when processing the FDs. This is due to the fact that each FD is processed only once. It does not require further processing. Thus, there is no need to add it to the active list.

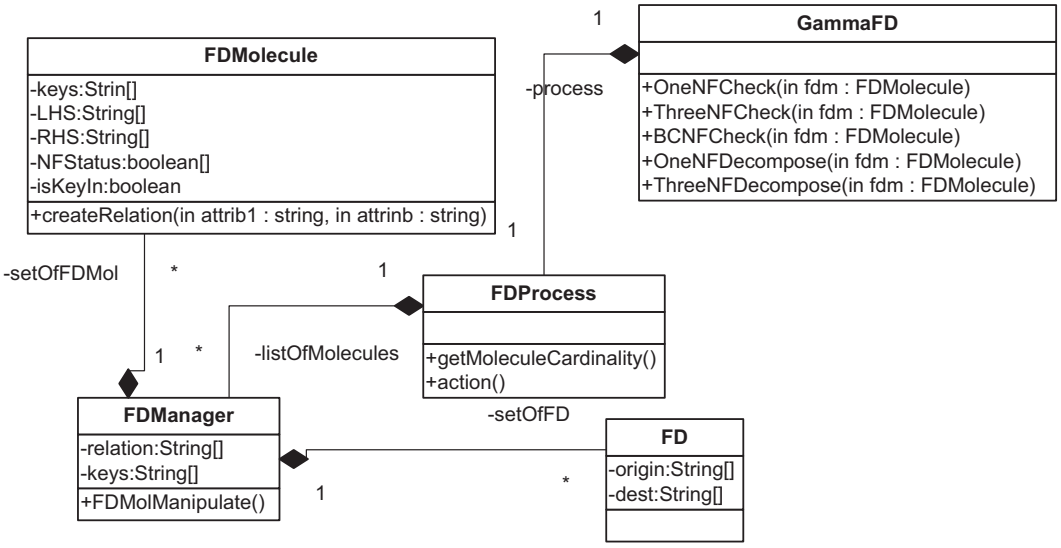

Fig. 3. The implementation framework of the data normalization on top of the GVM.

```
package services;
import java.util.LinkedList;
import tailor.Tailor;
public class GammaNorm extends Tailor{
    public int getMoleculeCardinality() {
    return 1;
    }
  public void action(LinkedList atoms, LinkedList
activeAtoms, LinkedList passiveAtoms) {
    FDMolecule fd;
    fd =(( FDMolecule) atoms.poll());
    if (! fd.getThreeNF()) {
        fd.setThreeNF(true);
        fd.createRelation(fd.getLHS(), fd.getRHS());
        fd.setIsKeyIn();
        fd.setKey(fd.getLHS());
    }
    passiveAtoms.add(fd);
    }
}
```

Fig. 4. A sample code implementing the action of the gamma program *3NFDecompose.*

6. Experimental Results

Several database relations with different sizes were tested for the conformance and the decomposition of the specified normal forms. Table 2 indicates the performance of GVM when using our Gamma specification as compared to sequential machines. The table highlights the result obtained for a database of 300 attributes. The number of workers was increased from 10 to 200 in an increment of 10. We report the performance of 10-100 and 200 workers. It can be seen from Table 2 and Fig. 5 that the GVM outperforms the sequential implementation superiorly. The improvement of the performance is due to exploiting the parallelism aspects in Gamma. It is observed that there is an optimal number of workers at which the Gamma machine performs best. This is dependent on the number of attributes being processed. In this case, 60 workers gave an optimal performance for 300 attributes. Increasing the number of workers degraded the performance as illustrated in Fig. 5. This is due to the higher communication time required among the workers. It should be noted that the testing was carried out on a machine with 2 GB of RAM, and a 4 GHz speed. Testing results may vary upon changing the hardware configurations. Compared to the work in (Touir *et al.*, 2008), this is a more up to date specification yielding better results.

The performance of the Gamma machine was examined with respect to the top-down and bottom-up approach. Relations with different number of attributes were given as input to the Gamma machine. Relations of up to 200 attributes were obtained from real life problems. Other relations were generated for the sake of testing the machine. The Gamma machine performed superiorly on the top-down approach as compared to the bottom-up approach as illustrated in Table 3 and Fig. 6. This is attributed to the extensive computation in the bottom-up approach as opposed to that in the top-down approach. This is also evident from the more elegant specification of the top-down approach given in Section 3 versus that of the bottom-up given in Section 4.

Table 2. Results of running the 3NF decomposition on the Gamma Machine vs. Sequential Machine

No. of Workers	Sequential Time	GM Time	Improvement
10	119	26	457%
20	119	17	700%
30	119	14	850%
40	119	10	1190%
50	119	7	1700%
60	119	6	1983%
70	119	7	1700%
80	119	8	1487%
90	119	8	1487%
100	119	11	1081%
200	119	16	743%

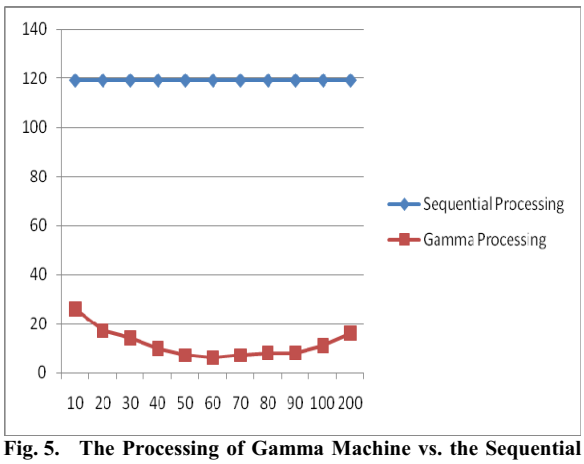

Fig. 5. The Processing of Gamma Machine vs. the Sequential Machine.

Furthermore, the process involved in the bottom-up approach requires more time in the communication among the workers of the Gamma machine. It is observed that when the number of workers increases, the computation time decreases to a minimal value then it starts increasing. This is due to the fact that the communication overhead and the synchronization between the different processes start consuming much more time than the execution of the intended tasks.

Table 3. Performance of Top-down approach vs. Bottom-up approach

Number of Attributes in R	GM Estimated Time		The Top-Down is consistently better
	Bottom-Up	Top-Down	
20	12	4	66.67%
50	27	5	68.75%
100	46	8	65.22%
200	61	13	64.86%
500	79	17	56.41%
1000	93	29	64.71%

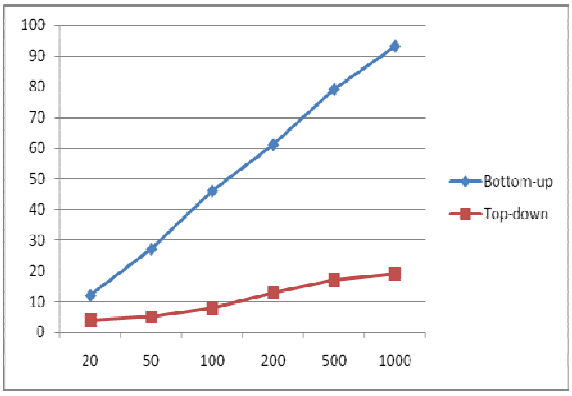

Fig. 6. Performance of the Top-down approach vs. the Bottom-up approach.

7. Conclusion

In this paper, we exploited the expressiveness of Gamma to elegantly and succinctly specify the normalization aspects in relational databases and use its computational power to achieve greater performance in verifying and realizing relational database normal forms. We presented succinct specifications of the normalization verification and realization in relational databases using Gamma through two approaches, namely, the top-down approach and the bottom-up approach. We report on the experimentation with an implementation of Gamma machine to demonstrate the computational gain of Gamma parallelism in solving useful problems with reasonably extensive computation. We believe this will facilitate the automatic database design that is based on sound principle. We are currently working on automating the complete process of the database design including generating applicable functional dependencies using Gamma. This requires augmenting the system with intelligence through the incorporation of knowledge bases. We are in the process of examining the application of Gamma in constructing and managing knowledge bases.

Acknowledgement. This work is partially supported by the Research Center of College of Computer and Information Sciences, King Saud University.

References

Akehurst, H.; Bordbar, D.; Rodgers, B. and Dalgliesh, N.T. "Automatic Normalization via Metamodeling." *Proc. of ASE 2002 Workshop on Declarative Meta Programming to Support Software Development*, Edinburgh, (2002).

Banâtre, J.-P.; Fradet, P. and Le Métayer, D. "Gamma and the Chemical Reaction Model: Fifteen Years After." *Workshop on Multiset Processing (WMP)*, Curtea de Arges, Romania, (2000), 17-44.

Berry, G. and Boudol, G. "The Chemical Abstract Machine." *Journal of Theoretical Computer Science*, Vol. 96, (1992), 217-248.

Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." *Comm. ACM*, Vol. 13, No. (6), (1970), 377-387.

Codd, E.F. "Further Normalization of the Data Base Relational Model." In: R. Rustin (Ed.), *Data Base Systems, Courant Computer Science Symposia*. Vol. 6, New York: Prentice-Hall, (May 1971).

Codd, E.F. "Recent Investigations into Relational Data Base Systems." IBM Research Report RJ1385 (April 23rd, 1974). Republished in *Proc. 1974 Congress*, Stockholm, Sweden, (1974).

Connolly, T. and Begg, C. *Database Systems: A Practical Approach to Design, Implementation, and Management*. 3rd ed., Reading, MA: Addison-Wesley, (2002).

Diederich, J. and Milton, J. "New Methods and Fast Algorithms for Database Normalization." *ACM TODS*, Vol. 13, No. (3), (1988), 339-365.

Elmasri, R. and Navathe, S.B. *Fundamentals of Database Systems*. 5th ed., Reading, MA: Addison-Wesley, (2007).

Inverardi, P. and Wolf, A. "Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model." *IEEE Transactions on Software Engineering*, Vol. 21, No. (4), (1995), 373-386.

Ling, T.; and Goh, C. "Logical Database Design with Inclusion Dependencies." *Proc. of the 8th Int. Conf. on Data Engineering*, Tempe, Arizona, USA, (1992), 642-649.

Maier, D. *The Theory of Relational Databases*. Rockville, MD: Computer Science Press, (1983).

Mentre, D.; Le Metayer, D. and Priol, T. "Formalization and Verification of Coherence Protocols with the Gamma Framework." *IEEE Software Engineering for Parallel and Distributed Systems*, Limerick, Ireland, (2000), 105-113.

Omiecinski, E. "A Parallel Algorithm for Relational Database Normalization." *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. (4), (1990), 415-423.

Touir, A.; Mathkour, H. and Al-Athel, D. "An Application of Gamma Formalism in Database Design." *Proceeding of the International Conference on Computer and Communication Engineering (ICCCE 2008)*, Kuala Lumpur, Malaysia, (2008), 974-977.

Wong, S.K.M.; Butz, C.J. and Xiang, Y. "Automated Database Schema Design using Mined Data Dependencies." *Journal of the American Society for Information Science*, Vol. 49, No. (5), (1998), 455-470.

ÅŋÅDZ ĞƗÅŎǩ ÿãÆǺǪèÅǣ ÅŦǪȞò ȸŋ ĘŎȞǪöãş ĞƗÛǣãÀǪöã ğÅφÅƗǤöã Æʒãʐę Ž ğÅφÅƗǤöã ʒƗǤÛǩ

øʐõØŋ ÖƗʒÅƧǕ ȸÊǵ

قسم علوم الحاسب ، كلية علوم الحاسب والمعلومات ،

جامعة الملك سعود ، الرياض ، المملكة العربية السعودية

mathkour@ccis.ksu.edu.sa, binmathkour@yahoo.com

(قدم للنشر في ٢١/١٢/٢٠٠٨م ؛ وقبل للنشر في ٢/٢/٢٠٠٩م)

ȬÅŸŋ ǪȞǤöã. يعتبر النموذج المجرد العام للتعامل مع المجموعات المتعددة (جاما) أسلوباً حسابياً وبرمجياً متوازياً يستخدم بنية المجموعات المتعددة والتي تعرف كثنائي من الشكل (الشرط ، الاستجابة). يتم التعامل مع عناصر المجموعات المتعددة في سلسلة من التفاعلات الكيميائية لإنتاج عناصر جديدة حسب مجموعة من الشروط. سوف نشرح في ورقة العمل هذه كيفية استخدام وتطبيق تقنية جاما والاستفادة من قوتها الحسابية لتوصيف وتكوين جداول بيانات مطبعة وفقاً لنماذج تطبيع البيانات الترابطية ، حيث يتسم التوصيف باستخدام تقنية جاما بالدقة والأناقة وتفادي التفاصيل التي قد تؤثر على صحة التوصيف. ونعرض في هذه الورقة طريقتين للتوصيف تستخدم في تصميم قواعد البيانات الترابطية ، ونناقش أداء بيئة عمل جاما في الحسابات المعقدة التي تتطلبها عمليات التطبيع ، كما نستعرض ونقارن أداء جاما مع طرق التنفيذ التسلسلية ، ونقارن أيضاً نتائج طريقتي التوصيف في بيئة جاما وذلك من خلال القيام بتجارب تستخدم فيها بيانات حقيقية.