

A SEMI-AUTOMATED PROCEDURE FOR  
SMALL RELATIONAL DATABASE  
LOGICAL DESIGN

By  
NAFIUR R KHANDAKER MOHAMMAD  
Bachelor of Science  
North South University  
Dhaka, Bangladesh  
1996

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
May, 2002

A SEMI-AUTOMATED PROCEDURE FOR  
SMALL RELATIONAL DATABASE  
LOGICAL DESIGN

Thesis Approved:

H. Lu

Thesis Adviser

D. E. Henderson

Paul Volpp III

Timothy J. Petruccio

Dean of the Graduate College

## ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Huizhu Lu for her careful supervision, constructive guidance, inspiration and friendship. My sincere appreciation extends to Dr. G. E. Hedrick and Dr. Nohphil Park whose guidance, assistance, encouragement, and friendship are also invaluable.

I would also like to give my special appreciation to my wife, Afsana Hamid, for her precious suggestions to my research, her strong encouragement at times of difficulty, love and understanding throughout my research endeavor. My earnest love and affection goes to my daughter Nuzhat Rahman, who was born during this paper write-up. I am grateful thanks to my parents Nadira Khandaker and Idris Ali Khandaker for their benevolence and blessings.

Finally, I would like to thank the members of the Department of Computer Science for their support during the two years of my study.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION .....	1
Relational database design.....	4
Requirements collection and analysis.....	5
Conceptual database design.....	7
Logical database design.....	9
Physical database design.....	9
Database authentication security .....	10
II. A SEMI-AUTOMATED PROCEDURE FOR SMALL RELATIONAL DATABASE LOGICAL DESIGN .....	12
Semi-automated logical database design procedure .....	13
Conventional database design vs. Semi-Automated database design.....	20
Authentication security model.....	21
III. IMPLEMENTATION AND RESULTS.....	24
Designing “ <i>Semi-Automated Database Design</i> ” software.....	24
Implementing “ <i>Semi-Automated Database Design</i> ” software.....	25
Implementing “ <i>Student Information System</i> ” software .....	26
IV. SUMMARY, CONCLUSIONS AND LIMITATIONS .....	31
REFERENCES .....	33
APPENDICES .....	35
APPENDIX A – IMPLEMENTATIONS.....	35

## LIST OF TABLES

Table	Page
I. Student Information System entity set and attribute set .....	16
II. Attribute mapping .....	18
III. Comparison between conventional and semi-automated database design .....	20
IV. Table definition of Student Information System .....	28

## LIST OF FIGURES

Figure	Page
1. Typical web database application structure .....	2
2. Information flow of database design phases .....	6
3. Semi-automated procedure for small relational database logical design .....	12
4. Student Information System DFD.....	15
5. Secure password based authentication model .....	22
6. “ <i>Semi-Automated Database Design</i> ” Software Structure .....	24
7. The relational database of Student Information System .....	27
8. Closure of a set of functional dependencies implementation.....	35
9. Canonical Cover of a set of functional dependencies .....	43
10. Dependency Preservation Test algorithm implementation .....	46
11. BCNF decomposition algorithm implementation .....	49
12. 3NF decomposition algorithm implementation .....	53
13. Basic project structure implementation.....	58
14. User interface to the Closure algorithm .....	62
15. User interface to the Canonical Cover algorithm.....	65
16. User interface to the Dependency Preservation Test .....	68
17. User interface to the Closer of Functional Dependency algorithm.....	71
18. User interface to the BCNF decomposition algorithm.....	74

Figure	Page
19. User interface to the semi-automated design decision algorithm .....	77
20. User interface for generic input form.....	80
21. User interface for project progress status.....	85
22. User interface to select a project type .....	87
23. User interface for 3NF decomposition algorithm .....	89
24. Application window for all the algorithms .....	92
25. SIS Instructor form.....	97
26. SIS Course form.....	97
27. SIS Student form.....	97
28. SIS Offer form.....	98
29. SIS Enrollment form .....	98
30. SIS Login screen .....	99
31. SIS Student home screen.....	99
32. SIS View student profile screen.....	100
33. SIS Update student profile screen .....	100
34. SIS login processor source code .....	101
35. SIS home page source code.....	101
36. SIS check login source code .....	102
37. SIS title bar source code.....	102
38. SIS status bar source code.....	102
39. SIS open database connection source code.....	102

Figure	Page
40. SIS close database connection source code .....	102
41. SIS view profile source code.....	103
42. SIS global.asa file.....	103

## NOMENCLATURE

ADO	ActiveX Data Objects
ASP	Active Server Page
DBMS	Database Management System
DDL	Data Definition Language
DML	Data Manipulation Language
FD	Functional Dependency
SIS	Student Information System
SQL	Structured Query Language

## CHAPTER I

### INTRODUCTION

Electronic Commerce (e-commerce) is one of the major driving powers of today's business economy. Business automation using internet information technology is the fundamental objective of e-commerce. Real-time business transactions, time and place independent information availability, consumer choice flexibility and customer information are a few dominant features of e-commerce. Web database application or internet enabled database application is the key component of e-commerce.

Database Systems have improved a great deal during the last few years to incorporate the ever-evolving internet technology. Significant improvements have been achieved to facilitate database access through internet. Specifically, server response time, remote connection management, web user interface development tools and data security are a few major development areas. The basic difference between a desktop database system and an web database system is data accessibility. Web database systems are accessible to anyone connected to the internet, on the other hand, desktop database systems are accessible to only those who have physical access to the Database Management System (DBMS).

Typical web database application consists of two basic layers, a front-end business logic layer and a back-end data storage layer. Business logic determines how

raw data is represented to various users. It also provides an appropriate user interface. Good examples of front-end business logic development tools are Active Server Pages and Java Server Pages. A database management system such as SQL Server or Oracle Server is used as back-end data storage to manage data efficiently and securely. Figure 1 shows the basic structure of a web database application.

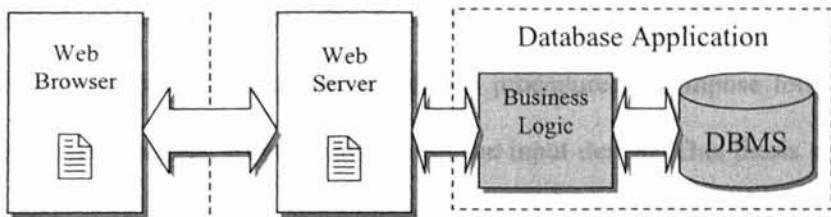


Figure 1: Typical web database application structure

Developing a complete web database application is a complex and sequential process. The start point of the development process is the back end database design. Due to the rapid growth of web database development, formal design processes such as normalization and integrity constraint checking are not always followed properly. This may lead to inconsistent data, jeopardizing the entire database development goal. Real-time data must be valid at any given point of time. Database System development to empower business requirements should not overlook data integrity. Ensuring consistent data through good database design and implementing a secure data access policy are two key requirements for successful business operation.

- Another aspect of on-line database system is data security. All database systems must protect sensitive business data (e.g. financial transactions) irrespective of their access technology. Secured and controlled data access is a must for web database system. Database security should not be compromised at any cost.

The objective of the thesis is to develop a semi-automated procedure for small relational database logical design. The procedure integrates several presently existing algorithms of normalization. The semi-automated procedure will impose formal design standards (e.g. third normal form or BCNF) on the input design. This thesis consists of four chapters.

- Chapter I provides an overview of the web database application structure, discusses the needs for formal database design methodologies and states the thesis objective. This chapter also includes a brief overview of literature for this thesis. The literature will help readers grasp conventional relational database design procedure.
- Chapter II discusses the semi-automated database design procedure, its advantages and disadvantages. This chapter also discusses an improved user authentication model suitable for web database application.
- Chapter III provides implementation details of the semi-automated design procedure and the architecture of the software. A complete web database application is implemented which incorporates and tests the methodologies developed in chapter II.

- Chapter IV summarizes this thesis work. It also discusses the limitations and suggests future work in this field.

The rest of this chapter is subdivided into two parts. The first part, Relational Database Design, focuses on the conventional database design techniques. The second part discusses Database Authentication Security, its use and improvement possibilities.

### 1.1 Relational Database Design

A group of tables and mapping cardinality among the tables is called a relational database. Each table in the database is a collection of attributes. Designing the logical structure of database based on the user requirement specification is the primary goal in relational database design. In other words, database design process takes user requirements as input and generates the actual table structures and relationship among tables as output. The user requirement specifies what the database system is supposed to do. The table structure specifies organization of data in a database by defining how the attributes are grouped together to form tables. Creating tables and establishing relationships among those tables as per standard design rules is the next logical step in database designing. Another important design goal is to make sure that the data stored in the database is not redundant and inconsistent. Redundant is repeating same information in more than one tables and redundancy is main source of inconsistency.

Elmasri and Navathe, (2000) have subdivided database design process into six basic phases:

1. Requirements collection and analysis
2. Conceptual database design
3. Choice of DBMS
4. Data model mapping (logical design)
5. Physical design
6. System implementation and tuning

Figure 2 shows information flow of each design phases.

#### 1.1.1 Requirements collection and analysis

Identifying and analyzing user needs related to the database system to be developed is known as Requirements Collection and Analysis. According to Elmasri and Navathe, (2000) the primary areas of requirements collection and analysis are

- a. identification of user groups and application areas
- b. review of existing systems
- c. analysis of the operating environment and
- d. identification of the processing requirements.

Requirements are described in natural language and mostly informal statements. These informal statements may be changed into a structured form by using one or more formal requirements specification techniques. Data Flow Diagrams (DFDs) and Flow Charts are

two frequently used diagrammatic methods for organizing and presenting information processing requirements.

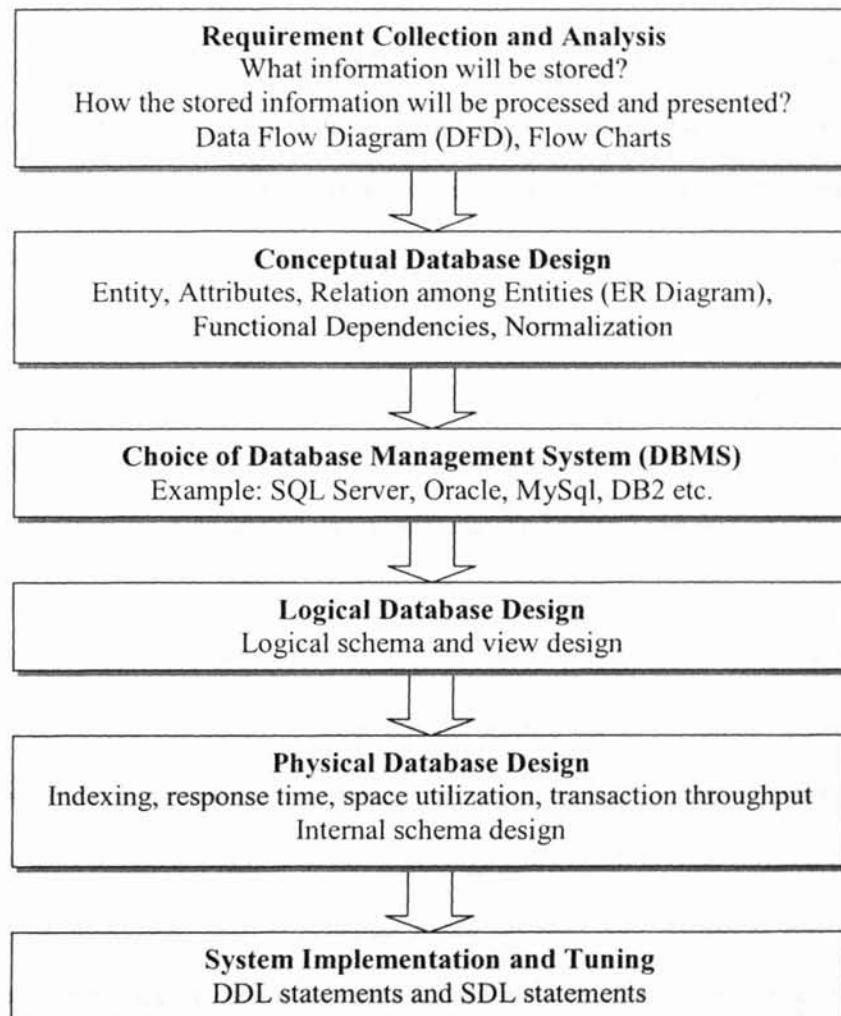


Figure 2: Information flow of database design phases  
(Elmasri and Navathe, 2000)

### 1.1.2 Conceptual database design

Conceptual database design examines the user data requirements and produces a conceptual schema in a DBMS independent high-level data model. Conceptual schema design is identifying entities and attributes, grouping attributes into schemas and finally deciding the mapping cardinalities among entities. Its goal is to generate a set of relational schemas that allows storing information without redundancy (Silberschatz, Korth and Sudarshan, 2001). The major challenge in database designing is modeling the real world in terms of table structure. Another consideration is data integrity. In addition to correct table structure, proper data integrity checking is needed to ensure information consistency. Repetition of information is not acceptable in schema design. High-level data model techniques such as the Entity Relationship (ER) model can be used to outline the database structure. Conceptual data model may be DBMS independent.

Given a problem definition, it is possible to design more than one relational schema. To identify the best relational schema from a set of relational schemas, Codd (1979a) proposed a normalization process. Normalization takes a relation schema through a series of tests to check whether or not it belongs to a certain normal form. It is a formal guideline for analyzing relation schemas based on their key attributes and functional dependencies among their attributes (Elmasri and Navathe, 2000). During normalization process relation schemas that do not meet the design standards are decomposed by breaking up into smaller relation schemas. Normalization test can be carried out on individual relation schemas separately. When a test fails, it must be decomposed into

more than one relation, each of which must meet the normalization test individually. According to Silberschatz, Korth and Sudarshan, (2001) desirable properties of schema decomposition are:

1. Lossless-join Decomposition
2. Dependency Preservation Decomposition

Integrity constraints are a set of constraints or rules associated with individual tables and attributes of the tables in the database. It ensures that add, edit and delete operations do not violate data consistency (Silberschatz, Korth and Sudarshan, 2001). Key attribute declarations and relationship constraint among tables are few means of formulating integrity constraints. In general, an integrity constraint can be an arbitrary restriction applied to the database. However, arbitrary restriction may be costly to test. Usually integrity constraints that can be tested easily are added directly to the database. Domain constraints are the very basic form of integrity constraints. It checks the value of an attribute against a given domain. Since bulk computation is not involved in the process of enforcing domain constraints, it can be tested easily by the system whenever a new data item is entered and updated in the database.

Referential integrity constraints ensure that - “a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation” (Silberschatz, Korth and Sudarshan, 2001). If one table does not contain the data, other table(s) should not. Some Relational Database Management Systems use their own built-in referential integrity checking to maintain data consistency (e.g. Microsoft Access). If it

is not implemented by the RDBMS, database system developer must provide integrity constraints checking with the system. This is an essential part of database system, so every database system must check referential integrity to make sure that every piece of data is consistent through out database system life cycle.

#### 1.1.3 Logical database design

The relation schemas generated in conceptual database design phase are mapped to a selected DBMS. Schema design generated in the conceptual design phase is converted to DBMS specific Data Definition Language (DDL) statements in this phase. Along with the table structure definition some extensions of DDL can be used to define referential integrity constraints among tables. This process is also known as data model mapping. Data Definition Language (DDL) is a part of Structured Query Language (SQL), used to create, alter and delete tables.

#### 1.1.4 Physical database design

Physical database design is the process of choosing specific data and file structures and access paths for the physical database files to achieve good performance for various database operations. According to Elmasri and Navathe, (2000) indexing, response time, space utilization, transaction throughput and clustering of related records on disk blocks are few essential criteria considered during physical database design

phase. Satisfying all these criteria will help achieving overall system performance requirements.

The data storage and access engine is a key component of any relational database. A successful database implementation requires careful planning of the physical design at the early stages of the project. The storage engine of a relational database requires much of this planning, which includes determining (MSDN Library, 2001):

- What type of disk hardware to use, such as RAID (redundant array of independent disks) devices?
- How to place data files onto the disks, such as Filegroups in SQL Server?
- Which index design to use to improve query performance in accessing data, such as Index Tuning Recommendations in SQL Server?
- How to set all configuration parameters appropriately for the database to perform well?

## 1.2 Database Authentication Security

Generally password-based authentication security or a login procedure is used to control access to the database server. A login procedure requests a user name and a password from the user. The authenticating operation matches the supplied user name and password with the password table (kept inside the server) to accept or reject the login request. Once the login request is granted, the user gains access to the database for further possible specific usage. The login procedure facilitates the server to identify the user.

Security checking after the login procedure may be described as access control security.

It determines whether a particular resource on the server is accessible to a user or not.

Login data (user name and password) should be secured at both the user end and the server end. Unfortunately, users do not always maintain password security, such as, sharing password among co-workers, writing it down, and choosing a small password are common security problems. The essential part of authentication security is to keep the password secured. Good password choice is the primary guard against hackers. If we can ensure password file security, we can theoretically guarantee the security of the authentication procedure. Almost all password-based authentication servers keep a copy of user name and password to validate login request. Servers must ensure the security of the password file. Modern operating systems do not store passwords directly; instead, they use an encryption/hashing algorithm to hide the actual password in the password file.

Web servers may have full access permissions granted to access the underlying database server, as a result, dynamic web applications have seamless access to the database server. The web server may represent a predefined authorized user to initiate a session with database server and provide access to different resources of the database. Database systems may have their own internal security measures.

The accessibility of each user may be defined separately for each individual table rather than full database server. This can be accomplished by an object based security policy provided by the database server (e.g. Microsoft Access 2000).

## A SEMI-AUTOMATED PROCEDURE FOR SMALL RELATIONAL DATABASE LOGICAL DESIGN

The starting point of database system development is requirement collection and analysis, and the final objective is to generate a set of schemas that are in appropriate normal form. The semi-automated procedure partially automates relation database logical schema design process. The procedure takes an initial schema design as input; decomposes it into several smaller schemas and finally produces Boyce-Codd Normal Form or Third Normal Form schema. Figure 3 shows the semi-automated procedure.

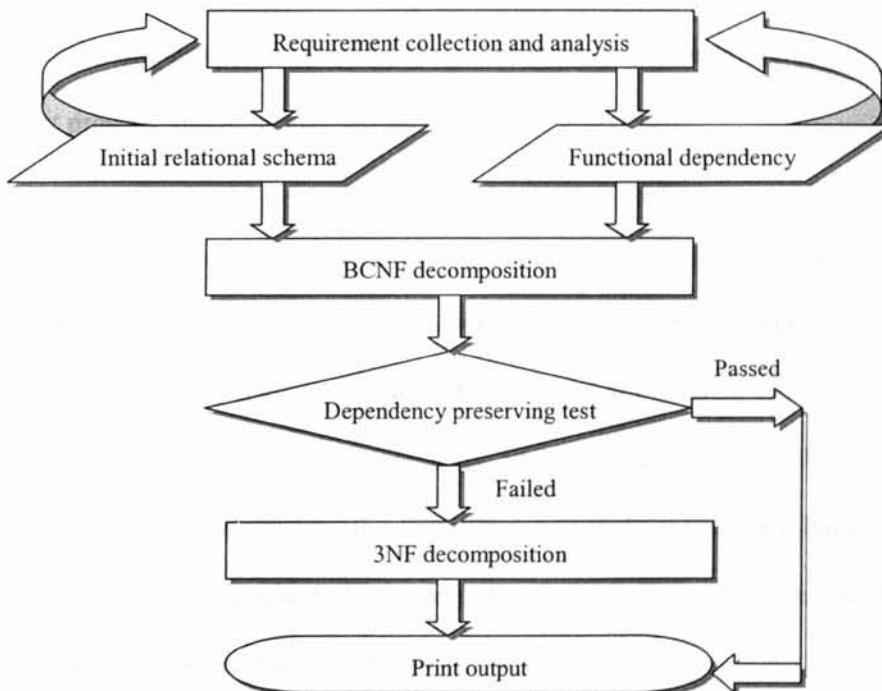


Figure 3: Semi-automated procedure for small relational database logical design

Before the detailed discussion of semi-automated procedure let us consider the following database design example problem, which is used to discuss each of the semi-automated logical design phases and their usage.

### 2.1.1 Example Problem Definition

Student Information System (SIS) is an online database application to manage and publish student grades. The primary objective of SIS is to provide grade monitoring functionalities to students and grade management, reporting and publishing functionalities to faculty members anytime anywhere. SIS facilitates students monitoring class performance closely and efficiently; moreover, it standardizes the grade management process for the faculty members.

Student grade is sensitive and the most valuable information to a college. System usage security (no one should be able to use the system other than students, instructors and record administrators) and grade publishing security (no student should be able to view other student's grade) are prime concerns. Students should not be able to change any data (e.g. grade) in the system other than their personal profile. As a basic security measure every user must login before using the system. Following are the detailed requirement specifications of the system:

- a. Every student has a unique id, name, email and start semester (when he/she started college education)
- b. Every course has a unique id, prefix, number and title.
- c. Every instructor has a unique id, name and email address.
- d. At the beginning of every semester a few courses are selected from the course list and one instructor is assigned to every selected course. The selected courses are called offered courses for that semester. Every offer has unique id, credit hours, days that class meets, One course may have more than one offering under different sections.
- e. Many students can enroll in one course offer and one student may enroll in more than one offer. Every enrollment is identified by a unique enrollment id and every enrollment results in a letter grade and pass-fail decision (as different department has different policy) at the end of semester.

#### 2.1.2 Semi-automated database design algorithm description

Semi-automated logical database design procedure is based on already existing normalization algorithms. There are several methods of normalization (e.g. Normalization using join dependencies, Domain key normal form, Normalization based on functional dependencies) (Silberschatz, Korth and Sudarshan, 2001). The author used normalization using functional dependencies. Following is the description of the database design procedure steps developed by the author:

1. Analyze requirements: Analyze all collected information to make sure that every requirement is listed properly. Data Flow Diagram can be developed at this stage. Complete requirements specification will be the output of this step.

Consider the example problem discussed in section 2.1.1. The problem definition does not discuss how the data will be processed and represented to the user. Data Flow Diagram (DFD) can be used to capture the data processing as data flows through the system. Figure 4 shows the DFD of the Student Information System for every semester.

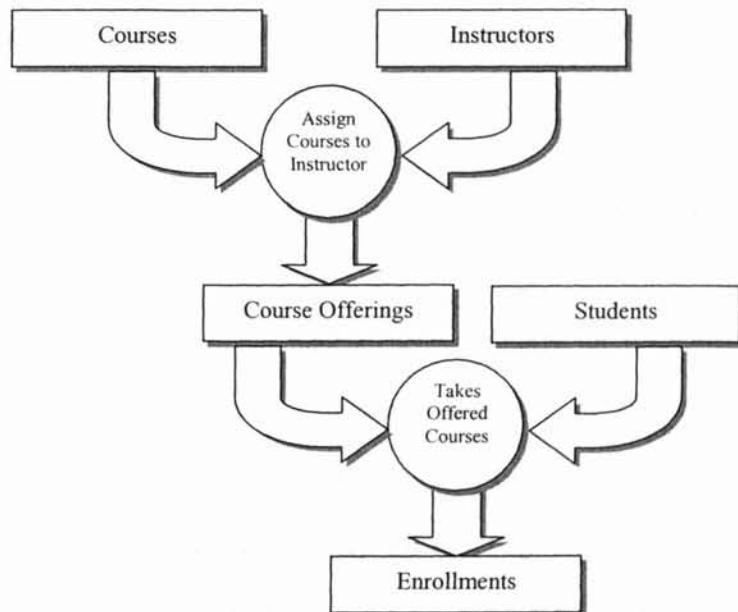


Figure 4: Student Information System DFD

2. Identify relation schemas: Identify and distinguish all relation schemas from the requirements specified in the previous step. Each relation schema is produced from an entity and a set of attributes associated with the entity. Different types of

attributes are also identified at this step (e.g. simple/composite, single/multi-valued, derived attribute). Identifying relation schemas and their attributes are iterative process. Before adding new schema every requirement specification must be analyzed properly. Relation schemas outlined at this step will be used as initial schema for the subsequent steps. Requirement specification analysis of SIS reveals the following entities and their attributes, listed in Table 1.

Entity	Attributes
Student	Id, Name, Email, Start Semester, Password
Course	Id, Title, Prefix, Number
Instructor	Id, Name, Email, Password
Offer	Id, Semester, Course Id, Instructor Id, Section, Credit Hour
Enrollment	Id, Offer Id, Student Id, Grade, Pass

Table 1: Student Information System entity set and attribute set

3. Identification of functional dependencies: Identify all functional dependencies that holds on each relation schemas identified in step 2. As defined by Silberschatz, Korth and Sudarshan, (2001) a functional dependency is a type of constraint that enforces the concept of key (e.g. primary key). It specifies the constraints on the set of legal relations. Functional dependencies allow us to express the facts about the tables that we are modeling in our database. The semi-automated design procedure finds the appropriate normal form of the input schema based on the functional dependencies identified in this phase. So it is very important that every functional dependency must be identified correctly.

Following are the list of functional dependencies that should hold on the Student Information System relational schemas.

- a. Student Id → Name, Phone, Email, Password
  - b. Instructor Id → Name, Email, Password
  - c. Course Id → Title, Prefix, Number
  - d. Offer Id → Semester, Section, Course Id, Instructor Id, Credit Hour
4. Application of BCNF decomposition algorithm: BCNF decomposition algorithm (Silberschatz, Korth and Sudarshan, 2001) is applied on the relation schemas determined in step 2. Functional dependencies identified in step 3 are used as parameters of the algorithm. The implementation developed by the author requires that every attribute should be mapped to a single character before using the algorithm. This specific mapping requirement reduces the amount of string comparison in the algorithm significantly as a result the algorithm runs comparatively faster. Table 2 shows attribute mapping of SIS.

Entity	Attribute	Representing Character
Student	Id	A
	Name	B
	Email	C
	Password	D
	Start Semester	E
Instructor	Id	F
	Name	G
	Email	H
	Password	I

Entity	Attribute	Representing Character
Course	Id	J
	Title	K
	Prefix	L
	Number	M
Offer	Id	N
	Semester	O
	Section	P
	Course Id	J
	Instructor Id	F
	Credit Hour	Q
Enrollment	Id	R
	Offer Id	N
	Student Id	A
	Grade	S
	Pass	T

Table 2: Attribute mapping

BCNF decomposition algorithm is applied on SIS as follows:

a. Input relation schema:

ABCDEFGHIJKLMNPQRST

b. Input functional dependencies:

i.  $A \rightarrow BCDE$

ii.  $F \rightarrow GHI$

iii.  $J \rightarrow KLM$

iv.  $N \rightarrow OPJFQ$

v.  $R \rightarrow NAST$

c. Output relation schemas in Boyce-Codd Normal Form:

- i. ABCDE
- ii. FGHI
- iii. JKLM
- iv. NOPJFQ
- v. RNAST

5. Checking dependency preservation: All BCNF decompositions may not be dependency preserving. Dependency preserving test algorithm is applied to the output of step 4. If all the relation schemas are dependency preserving, then skip step 6 and go to step 7. The output schema of this step will be BCNF, lossless-join and dependency preserving. If the dependency preserving test fails, go to step 6. SIS relation schemas generated in step 5 are all dependency preserving.
6. Application of 3NF decomposition algorithm: Third Normal Form decomposition algorithm is applied on relation schemas determined in step 2. This step is executed only when the output schema of step 4 is not dependency preserving. The output schema of this step will be 3NF, lossless-join and dependency preserving.
7. Print output: Final relation schemas are printed at this step.

## 2.2 Conventional Database Design vs. Semi-Automated Database Design

The following table summarizes the comparison between conventional and semi-automated database design process:

Conventional Database Design	Semi-Automated Database Design
<b>Manual process:</b> Schema in appropriate normal form is computed manually.	<b>Semi-Automated process:</b> Schema in appropriate normal form is computed by machine.
Requires knowledge, experience, and analytical capabilities.	Once the functional dependencies are determined, time complexity to compute closer of the functional dependencies is exponential.

Table 3: Comparison between conventional and semi-automated database design

### 2.3 Authentication Security Model

Simplicity and wide usability are the two major advantages of password-based authentication system. Almost all Operating Systems and server applications use some form of login procedure to restrict access to resources. Though the basic login data requirements of user id and password are the same, details of login validation procedure and password table management vary a great deal. Login security depends on how well the password is secured. For typical web application password based authentication security is the most popular form of security measure.

Web applications use TCP/IP protocol for communication between client and server. Due to the inherent nature of the IP protocol (data from client to server may be transported through a number of intermediate computers), web application authentication model should hide actual password during authentication process. Message Digest algorithm is a well-known one-way hash function developed by RSA Data Security, Inc. and MIT Laboratory. It takes an input message of arbitrary length and generates a 128-bit "fingerprint" or "message digest" as output. It is computationally infeasible to reverse the fingerprint generated by the algorithm. Moreover two input string cannot have the same message digest (fingerprint). The algorithm performs a series of basic mathematical operations on the input string. Since the algorithm is composed of basic mathematical operations it can easily be transported and executed on client computer. The Message Digest algorithm version five is the latest and it is an extension of the earlier version four.

Message Digest algorithm is used to protect the actual password during transportation from client to server. Instead of transporting actual password a fingerprint is generated and transported from the client machine to server machine. Figure 5 shows a secure password based authentication model.

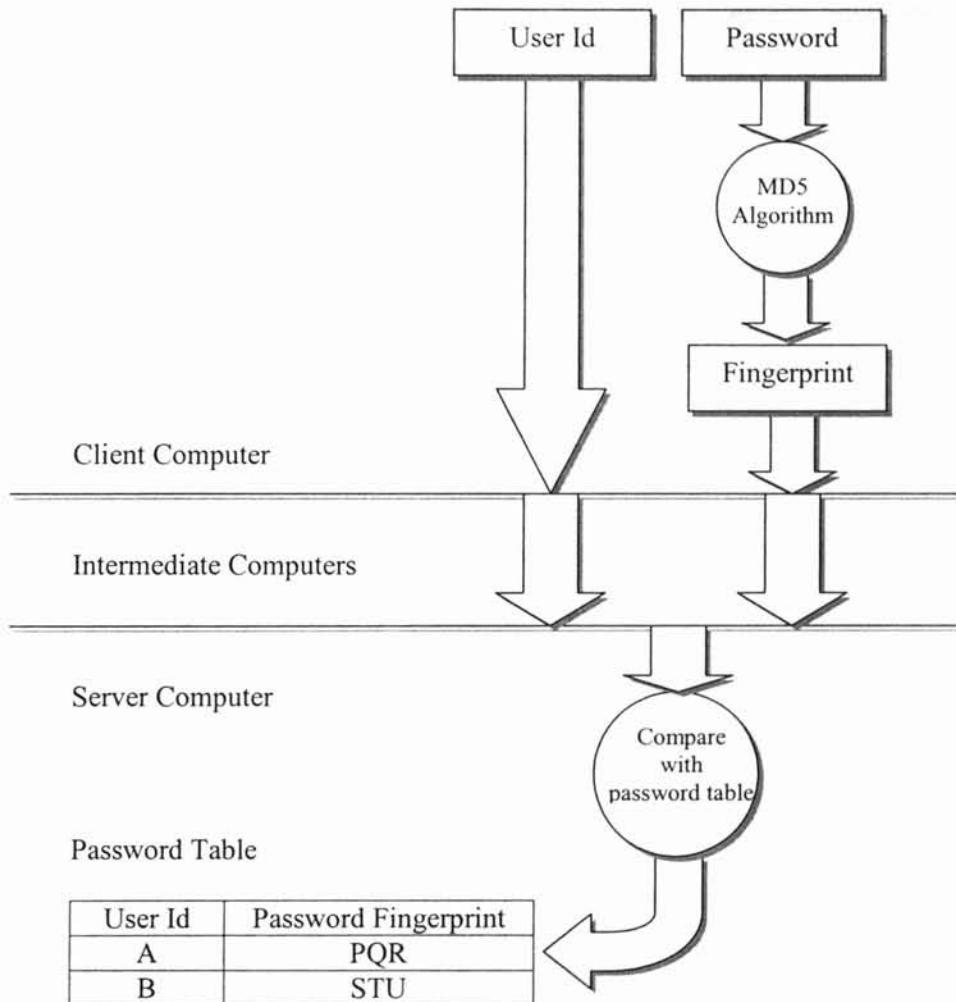


Figure 5: Secure password based authentication model

Advantages of using authentication security model based on Message Digest algorithm are as follows:

- a. Password is not transferred from client to server in plain text format
- b. Password is not stored in the password table in a plain text format
- c. Even if some one finds out the password fingerprint. He/she will not be able to login since the client login process will always compute fingerprint of the supplied password before sending it to the server process.
- d. Given a fingerprint it is computationally infeasible to find the actual password.

## IMPLEMENTATION AND RESULTS

“*Semi-Automated Database Design*” software is a direct implementation of the semi-automated logical database design procedure discussed in chapter II. This software is a general-purpose design tool that can be used to design small relational database schema for a given set of functional dependencies.

### 3.1 Designing “*Semi-Automated Database Design*” software

“*Semi-Automated Database Design*” software has three basic components: a user interface, an algorithm manager and an integration manager. The user interface provides an appropriate graphical user interface to start and operate the algorithms. The algorithm manager holds a collection of algorithms. The integration manager facilitates interoperability among algorithms. Figure 6 shows the basic structure of the software.

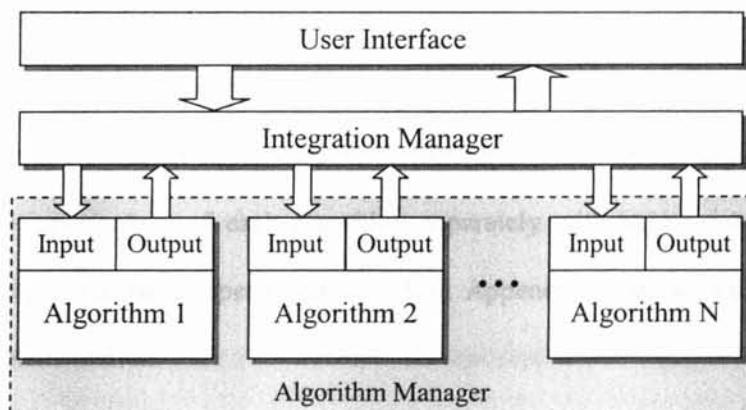


Figure 6: “*Semi-Automated Database Design*” Software Structure

Implementation of “*Semi-Automated Database Design*” software is accomplished in three phases: implementing the core algorithms, integrating the algorithms and implementing the user interface. “Java” has been chosen as the implementation language due to its inherent nature of platform independence, reusability and GUI development facilities.

### 3.2.1 Implementing the core algorithms

The “*Semi-Automated Database Design*” software consists of five basic algorithms: closure of a set of functional dependencies, canonical cover for a set of functional dependencies, testing for dependency preservation, Boyce-Codd normal form decomposition and third normal form decomposition. Algorithm implementations are shown in Appendix A (figure 8, 9, 10, 11 and 12).

### 3.2.2 Integrating the algorithms

Integration manager is responsible for interoperability among algorithms. It also manages the input/output of each algorithm separately with appropriate data type to improve overall processing speed. Figure 13 in Appendix A shows the basic project structure implementation.

### 3.2.3 Developing user interface

Graphical user interface helps operating the software efficiently. The software is able to manage each design problem as a separate project. The project management can further be subdivided into several smaller steps. Each step of the project corresponds to a single algorithm. User interface also facilitates the use of each algorithm separately. Figure 14 through 24 in Appendix A shows various sections of user interface implementation.

## 3.3 Implementing “*Student Information System*” software

Chapter II discussed the conceptual design of “*Student Information System*”. This section discusses logical design, physical design, implementation and finally operations details of SIS.

### 3.3.1 Logical Database Design of SIS:

Logical Database Design is defining the table structure based on the E-R diagram. The choice of data type and data width is entirely on the system developer. These structure definitions can be used by Data Definition Language (DDL) to create the actual table in a selected DBMS. Figure 7 shows the relational database of SIS and Table 4 shows the details of each table structure. MS Access supports creating tables in structure design view directly from the information provided in Table 4 without using DDL.

Figure 7: The relational database of Student Information System

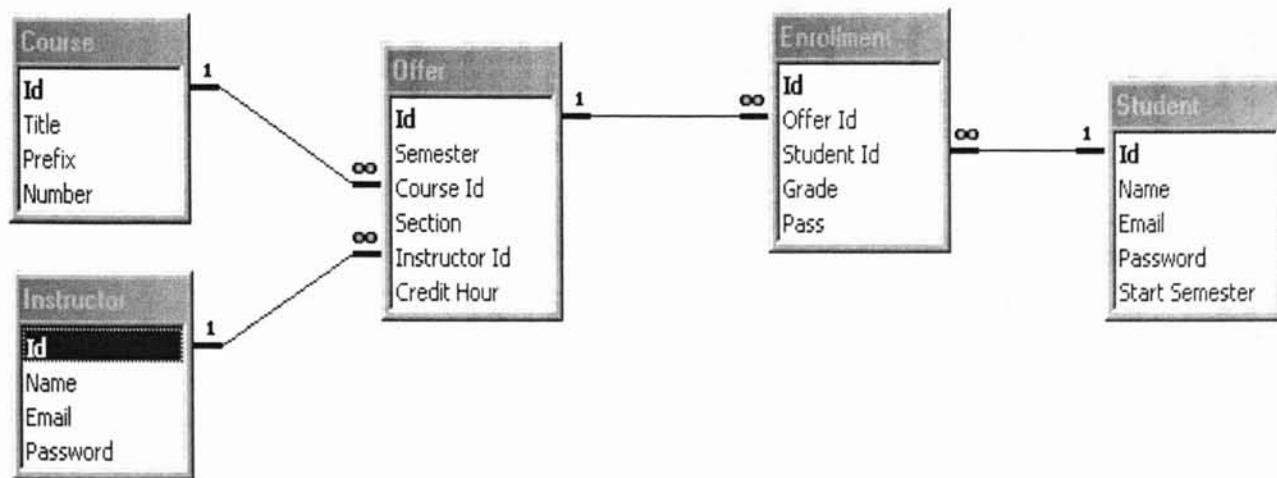


Table: Course

Name	Type	Size	PK/FK
Id	Long Integer	4	PK
Title	Text	50	--
Prefix	Text	3	--
Number	Text	6	--

Table: Instructor

Name	Type	Size	PK/FK
Id	Long Integer	4	PK
Name	Text	50	--
Email	Text	50	--
Password	Text	20	--

Table: Student

Name	Type	Size	PK/FK
Id	Text	10	PK
Name	Text	50	--
Email	Text	50	--
Start Semester	Text	15	--
Password	Text	20	--

Table: Offer

Name	Type	Size	PK/FK
Id	Long Integer	4	PK
Semester	Text	15	--
Course Id	Long Integer	4	FK
Section	Byte	1	--
Instructor Id	Long Integer	4	FK
Credit Hour	Byte	1	--

Table: Enrollment

Name	Type	Size	PK/FK
Id	Long Integer	4	PK
Offer Id	Long Integer	4	FK
Student Id	Text	10	FK
Grade	Text	1	--
Pass	Yes/No	1	--

Table 4: Table definition of Student Information System

### 3.3.2 Physical Database Design

The DBMS chosen for this example problem is Microsoft Access 2000. Physical database design issues such as index structure, file organization and configuration parameters are set by MS Access as default.

### 3.3.3 Database Authentication Security

User login procedure is used to ensure that no one other than students, instructors and record administrators use the system. Every login request is verified against the password fingerprint kept in the table. After successful login, user id (e.g. student id, instructor id) field is used to filter any information sent to a user. This filtering process ensures that no user receives other user's data.

### 3.3.4 System Operations of SIS

As stated in the requirements specification, there are two types of primary users - students and instructors. Instructors produce and populate data into the system and students view data and monitor their class performance. At the beginning of every semester all previous system data are backed-up and the system is prepared for the new semester. The system preparation takes place by adding all new instructors (see figure 25), new courses (see figure 26), new students (see figure 27), new offers (see figure 28), enrollments (see figure 29) and setting the current semester (e.g. Fall2001) as system parameter. At the end of the semester a letter grade along with a pass/fail decision is

assigned to every enrolled student under every offer. Students first login to the system (see figure 30) and then check their grades (see figure 31). Students can view and change their profile (see figure 32, 33). Figure 34 to 42 shows some selected source code of SIS.

## CHAPTER IV

### SUMMARY, CONCLUSIONS AND LIMITATIONS

In this thesis a semi-automated logical database design procedure is developed, implemented and tested. The semi-automated procedure combines several already existing database design algorithms into a single sequential algorithm. It is implemented as a software package named “*Semi-Automated Database Design*”. The software can be used for three different purposes:

- a. Normalize a given set of relational schema under a set of FD's to an appropriate normal form
- b. Test a relation schema for appropriate normal form
- c. Apply database design algorithms individually

An improved security model for database user authentication is developed and tested. The model uses Message Digest 5 algorithm to hide password transportation from client to server. The model does not require storing plain text password in the password table.

A part of example problem (*Student Information System*) is used to test the “*Semi-Automated Database Design*” software and the authentication security model. The complete relational database system “*Student Information System*” is implemented using Microsoft Access 2000. The web-interface is developed utilizing Active Server Page, VBScript, JavaScript and ActiveX Data Objects (ADO) technology.

The “*Semi-Automated Database Design*” software is partially automated as the Functional Dependencies are identified manually (human errors may exist). Referential Integrity is not provided by the procedure. The 1<sup>st</sup> and 2<sup>nd</sup> normal forms are not included in the procedure. The size of schema is limited to a small number due to the time complexity and space complexity.

The computation of the closure of the given set of functional dependencies used in the semi-automated procedure takes exponential time. Minor increase in the input schema size increases the computation time significantly. This is the major disadvantage of the “*Semi-Automated Database Design*” software. Moreover the software requires that every attribute must be represented by one single character to reduce string comparison time.

## REFERENCES

1. Ayers, Danny, Bergsten, Hans, Bogovich, Michael, Diamond, Jason, Ferris, Matthew, Fleury, Marc, Halberstadt, Ari, Houle, Paul, Mohseni, Piroz, Patzer, Andrew, Phillips, Ron, Li, Sing, Vedati, Krishna, Wilcox, Mark and Zeiger, Stefan (1999). Professional Java Server Programming. Chicago, Illinois: Wrox Press.
2. Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks.", Communications of the ACM, Vol. 13, No. 6, pp. 377-387.
3. Codd, E. F. (1979) "Extending the Data Base Relational Model to Capture More Meaning.", ACM Trans. on Database Systems, Vol. 4, No. 4, pp. 397-434.
4. Date, C. J. (2000). An Introduction to Database Systems. 7<sup>th</sup> Ed. New York, NY: Addison Wesley Publishing Company, Inc.
5. DevEdge Online Documentation (1997). "Introduction to SSL.", <http://developer.netscape.com/docs/manuals/security/sslin/contents.htm>, Netscape Communications Corporation
6. DevEdge Online Documentation (1997). "Introduction to Public-key Cryptography." <http://developer.netscape.com/docs/manuals/security/pkin/index.htm>, Netscape Communications Corporation
7. Elmasri, Ramez and Navathe, Shamkant B. (2000). Fundamentals of Database Systems. 3<sup>rd</sup> Ed. New York, NY: Addison Wesley Publishing Company, Inc.
8. Haecke, Bernard Van (1997). The Power Guide to Integrating Enterprise Databases with Web Applications JDBC: Java Database Connectivity. IDG Books.
9. Krochmal, Mo. "E-Commerce Definition" <http://www.techweb.com>, TechWeb News, 1999.
10. Litwin, Paul, Getz, Ken, and Reddick, Greg. "Fundamentals of Relational Database Design." <http://www.microsoft.com>, MSDN Library, 1989.
11. Rahmel, Dan. "Database Security." <http://www.dbmsmag.com>, Internet Systems, 1999.
12. Rivest, R. (1992). The MD5 Message-Digest Algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc.

13. Rob, Peter and Coronel, Carlos (1995). Database Systems Design, Implementation, and Management. Boyd & Fraser
14. Silberschatz, Abraham, Korth, Henry F. and Sudarshan, S. (2001). Database System Concepts. 4<sup>th</sup> Ed. New York, NY: McGraw Hill.

## APPENDIX A

### IMPLEMENTATIONS

```
*****
File: Closure.java
LastUpdate: July 16, 2001
Usage: Compute closure of a set of Functional Dependencies
*****/

package Project.Algorithms;

import java.util.*;
import java.io.*;
import Project.ProgressStatus;

// ****
// class definition
public class Closure{

    // components space
    private TreeSet comRef, comAug, comTran;

    // attribute list
    private TreeSet attrList;

    // attribute subset list
    private TreeSet attrSubset;
    private Object[] attrSet;
    private boolean[] attr;

    // ****
    // This function computes closure of a given set of function dependencies
    // and a relation schema
    // Reference:
    // Silberschatz, Korth, Sudarshan; Database System Concepts,
    // 3rd Edition, Section: 6.5.2
    public TreeSet compile( TreeSet inputProd, TreeSet inputAttr ){

        // popup the progress bar dialog
        ProgressStatus.startProgressBar( "Compiling - Please wait" );

        // temporary working space
        TreeSet prod;

        // number of productions in the output production list
        int prodCount = 0;

        // output production space
        TreeSet outputProd = new TreeSet( inputProd );

        // get attributes from the input productions
        attrList = attributes( inputProd );
        // add input relation schema list
        attributes( inputAttr, attrList );

        // compute all possible subsets of the attribute list
        attrSet = attrList.toArray();
        attrSubset = new TreeSet();
```

```

        attr = new boolean[ attrSet.length + 1 ];
        Subset( attrSet.length );

        // initialize all component space
        comRef = new TreeSet();
        comAug = new TreeSet();
        comTran = new TreeSet();

        do{
            // save current production list count
            prodCount = outputProd.size();

            // apply reflexivity rule
            prod = reflexivity( outputProd, attrSubset );
            comRef.addAll( prod );
            outputProd.addAll( prod );

            // apply augmentation rule
            prod = augmentation( outputProd, attrSubset );
            comAug.addAll( prod );
            outputProd.addAll( prod );

            // apply transitivity rule
            prod = transitivity( outputProd );
            comTran.addAll( prod );
            outputProd.addAll( prod );

            // continue till new productions are added
            }while( outputProd.size() > prodCount );

            ProgressStatus.stopProgressBar();

            // return closure of the input production list
            return outputProd;
        }

        // *****
        // This function computes closure of a given set of function dependencies
        // Reference:
        // Silberschatz, Korth, Sudarshan; Database System Concepts,
        // 3rd Edition, Section: 6.5.2
        public TreeSet compile( TreeSet inputProd ){

            // temporary working space
            TreeSet prod;

            // number of productions in the output production list
            int prodCount = 0;

            // output production space
            TreeSet outputProd = new TreeSet( inputProd );

            // get all attributes from productions
            attrList = attributes( inputProd );

            // compute all possible subsets of the attribute set
            attrSet = attrList.toArray();
            attrSubset = new TreeSet();
            attr = new boolean[ attrSet.length + 1 ];
            Subset( attrSet.length );

            // initialize all component space
            comRef = new TreeSet();
            comAug = new TreeSet();
            comTran = new TreeSet();

            do{
                // save current production count
                prodCount = outputProd.size();

```

```

        // apply reflexivity rule
        prod = reflexivity( outputProd, attrSubset );
        comRef.addAll( prod );
        outputProd.addAll( prod );

        // apply augmentation rule
        prod = augmentation( outputProd, attrSubset );
        comAug.addAll( prod );
        outputProd.addAll( prod );

        // apply transitivity rule
        prod = transitivity( outputProd );
        comTran.addAll( prod );
        outputProd.addAll( prod );

        // continue till new productions are added
    }while( outputProd.size() > prodCount );

        // return closure of the input production list
    return outputProd;
}

// *****
// This function applies the reflexivity rule on a set of productions
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.2
private TreeSet reflexivity( TreeSet prodList, TreeSet attrSubset ){

    // update progress bar text
    ProgressStatus.progressText("Reflexivity...");

    // local working variables
    Iterator alpha, beta;
    String aStr, bStr, newProd;
    TreeSet newProdList = new TreeSet();

    // for all subset of attribute list
    alpha = attrSubset.iterator();
    while( alpha.hasNext() ){
        // take one subset
        aStr = (String) alpha.next();

        // for all subset of attribute list
        beta = attrSubset.iterator();
        while( beta.hasNext() ){
            // take another subset
            bStr = (String) beta.next();

            // new production
            newProd = aStr + "-" + bStr;

            // check if the right side is the subset of left side and
            // the production already exists in the list
            if( subsetOf( aStr, bStr ) == true &&
                prodList.contains( newProd ) == false ){

                // add the production to the list
                newProdList.add( aStr + "-" + bStr );

                // update the prgrssss bar
                ProgressStatus.progressBar();
            }
        }
    }

    return newProdList;
}

```

```

// *****
// This function applies the augmentation rule on the given set of
// functional dependencies
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.2
private TreeSet augmentation( TreeSet prodList, TreeSet attrSubset ){

    // update progress bar text
    ProgressStatus.progressText("Augmentation...");

    // local working variables
    Iterator alpha, beta;
    String prodL, prodR, prod, attr, newProd, newProdL, newProdR;
    TreeSet newProdList = new TreeSet();

    // for all productions
    alpha = prodList.iterator();
    while( alpha.hasNext() ){
        // take one production
        prod = (String) alpha.next();
        // take left side
        prodL = prod.substring(0, prod.indexOf('-'));
        // take right side
        prodR = prod.substring( prod.indexOf('-')+1, prod.length());

        // for all subsets of attribute list
        beta = attrSubset.iterator();
        while( beta.hasNext() ){
            // take one subset from the list
            attr = (String) beta.next();
            // compute union of the left side of the production and
            // the attribute subset
            newProdL = stringUnion( prodL, attr );
            // compute union of the right side of the production and
            // the attribute subset
            newProdR = stringUnion( prodR, attr );
            // form a new production
            newProd = newProdL + " - " + newProdR;

            // if the new production is not already in the list
            if( prodList.contains( newProd ) == false ){
                // add the production
                newProdList.add( newProd );

                // update progress bar
                ProgressStatus.progressBar();
            }
        }
    }

    return newProdList;
}

// *****
// This function applies the transitivity rule on a given set of
// functional dependencies
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.2
private TreeSet transitivity( TreeSet prodList ){

    // update progress bar text
    ProgressStatus.progressText("Transitivity...");

    // local working variables
    Iterator alpha, beta;
    TreeSet newProdList = new TreeSet();
    String pA, pB, aL, aR, bL, bR, newP;
}

```

```

// for all productions in the list
alpha = prodList.iterator();
while( alpha.hasNext() ){
    // take one production
    pA = (String) alpha.next();
    // take left side
    aL = pA.substring(0, pA.indexOf('-'));
    // take right side
    aR = pA.substring( pA.indexOf('-')+1, pA.length());

    // for all productions in the list
    beta = prodList.iterator();
    while( beta.hasNext() ){
        // take another production
        pB = (String) beta.next();
        // take left side
        bL = pB.substring(0, pB.indexOf('-'));
        // take right side
        bR = pB.substring( pB.indexOf('-')+1, pB.length());
        // form a new production from left side of pA and right
        // side of pB
        newP = aL + "-" + bR;

        // if the right side of pA equals left side of pB and the
        // list does not contain new production
        if( aR.equals( bL ) && prodList.contains(newP) == false){
            // add the production to the list
            newProdList.add( newP );

            // update progress bar
            ProgressStatus.progressBar();
        }
    }
}

return newProdList;
}

// *****
// This function gets all attributes from the input productions list and
// already existing attribute list
public static void attributes( TreeSet prodList, TreeSet aList ){

    // local working variables
    Iterator prodIterator;
    String prod;
    int prodLen;
    char attr;

    // for all productions
    prodIterator = prodList.iterator();
    while( prodIterator.hasNext() ){

        // take one production
        prod = (String) prodIterator.next();

        // get length of the production
        prodLen = prod.trim().length();

        // for all attributes in the production
        for( int j=0; j < prodLen; j++ ){

            // take one attribute
            attr = prod.charAt( j );

            // check if it is attribute
            if( Character.isLetterOrDigit( attr ) ){

                // add to the attribute list
                aList.add( new Character( attr ) );
            }
        }
    }
}

```

```

        }
    }

// *****
// This function gets all attribute from the input productions list
public static TreeSet attributes( TreeSet prodList ){

    // local working variables
    TreeSet aList;
    Iterator prodIterator;
    String prod;
    int prodLen;
    char attr;

    // new attribute list
    aList = new TreeSet();

    // for all productions
    prodIterator = prodList.iterator();
    while( prodIterator.hasNext() ){

        // take one production
        prod = (String) prodIterator.next();

        // get length of the production
        prodLen = prod.trim().length();

        // for all attributes in the production
        for( int j=0; j < prodLen; j++ ){

            // take one attribute
            attr = prod.charAt( j );

            // check if its a attribute
            if( Character.isLetterOrDigit( attr ) ){

                // add to the attribute list
                aList.add( new Character( attr ) );
            }
        }
    }

    // return attribute list
    return aList;
}

// *****
// This function generates all possible subsets of the attribute list
private void Subset( int n ){

    if( n == 0 ){
        writeSubset();
    }else{
        attr[ n ] = false; Subset( n-1 );
        attr[ n ] = true; Subset( n-1 );
    }
}

// *****
// This function writes subset computed by Subset function
private void writeSubset(){

    StringBuffer strBuff = new StringBuffer();

    try{
        // compute the subset
        for( int i=1; i <= attrSet.length; i++ ){
            if( attr[i] == true ){

```

```

                strBuff.append( attrSet[i-1] );
            }

            // add the subset to the subset list
            if( strBuff.length() > 0 ){
                attrSubset.add( strBuff.toString() );
            }
        }catch( Exception err ){
            System.out.println("writeSubset: " + err );
        }
    }

    // *****
    // This function tests if strB is subset of strA
    public static boolean subsetOf( String strA, String strB ){
        int lenB = strB.length();
        char chB;

        for( int i = 0; i < lenB; i++ ){
            chB = strB.charAt(i);

            if( strA.indexOf( chB ) == -1 ){
                return false;
            }
        }

        return true;
    }

    // *****
    // This function computes union of two strings
    public static String stringUnion( String strA, String strB ){
        StringBuffer strC = new StringBuffer();
        char chA, chB;
        int lenA = strA.length();
        int lenB = strB.length();
        int a=0, b=0, c=0;

        // continue till the end of either string
        while( a < lenA && b < lenB ){
            // union char
            chA = strA.charAt( a );
            chB = strB.charAt( b );

            if( chA < chB ){
                strC.append( chA );
                a++;
            }
            else if( chA > chB ){
                strC.append( chB );
                b++;
            }
            else{
                strC.append( chA );
                a++; b++;
            }
        }

        // add all the left over chars of string A
        while( a < lenA ){
            chA = strA.charAt( a );
            strC.append( chA );
            a++;
        }

        // add all the left over chars of string B
        while( b < lenB ){
            chB = strB.charAt( b );
            strC.append( chB );
            b++;
        }
    }
}

```

```

        strC.append( chB );
        b++;
    }

    return strC.toString();
}

// *****
// This function converts a TreeSet data structure to a string
private String TreeSetToString( TreeSet set ){
    StringBuffer buff = new StringBuffer();

    // if the given set is not null
    if( set != null ){
        // for all elements of the set
        Iterator i = set.iterator();
        while( i.hasNext() ){
            // add element to the string buffer
            buff.append( (String) i.next() + "\n" );
        }
    }

    return buff.toString();
}

// *****
// This function returns only the productions added to the closure list
// by Reflexivity rule
public String getReflexivity(){
    return TreeSetToString( comRef );
}

// *****
// This function returns only the productions added to the closure list
// by Augmentation rule
public String getAugmentation(){
    return TreeSetToString( comAug );
}

// *****
// This function returns only the productions added to the closure list
// by Transitivity rule
public String getTransitivity(){
    return TreeSetToString( comTran );
}

// *****
// This function returns the attribute list used for the closure
// algorithm
public String getAttributes(){
    StringBuffer buff = new StringBuffer();

    // if the attribute list is not null
    if( attrList != null ){
        // for all attribute of the list
        Iterator i = attrList.iterator();
        while( i.hasNext() ){
            // add each attribute to the string
            buff.append( (Character) i.next() + "\n" );
        }
    }

    return buff.toString();
}

// *****

```

```

    // This function returns all attribute subset used by the
    // closure algorithm
    public String getAttributeSubset(){
        return TreeSetToString( attrSubset );
    }
}

```

Figure 8: Closure of a set of functional dependencies implementation

```

/*
File: Canonical.java
LastUpdate: July 16, 2001
Usage: Compute canonical cover of a a set of functional dependencies
***** */

package Project.Algorithms;

import java.util.*;
import java.io.*;
import Project.Algorithms.Closure;
import Project.ProgressStatus;

// ****
// class definition
public class Canonical{
    // closure algorithm
    private Closure closure;

    // ****
    // constructor
    public Canonical(){
        // instantiate new closure algorithm
        closure = new Closure();
    }

    // ****
    // This function computes the canonical cover of a given set of
    // functional dependencies
    // Reference:
    // Silberschatz, Korth, Sudarshan; Database System Concepts,
    // 3rd Edition, Section: 6.5.4
    public TreeSet compile( TreeSet prodList ){

        // start progress bar
        ProgressStatus.startProgressBar("Compiling - Please wait");

        // local working variables
        TreeSet oldProdList = new TreeSet();
        TreeSet newProdList = new TreeSet( prodList );

        do{
            // save all productions
            oldProdList.clear();
            oldProdList.addAll(newProdList);

            // apply union rule
            newProdList = unionRule( newProdList );

            // remove all extraneous attributes
            newProdList = removeExtraneous( newProdList );

        // continue till the production list changes
    }
}

```

```

        }while( !newProdList.equals( oldProdList ) );

        // stop the progress bar
        ProgressStatus.stopProgressBar();

        return newProdList;
    }

    // *****
    // This function applies the union rule on the given set of productions
    // Reference:
    // Silberschatz, Korth, Sudarshan; Database System Concepts,
    // 3rd Edition, Section: 6.5.4
    private TreeSet unionRule( TreeSet prodList ){

        // update progress bar text
        ProgressStatus.progressText("Applying union rule...");

        // local working variables
        Iterator alpha, beta;
        TreeSet newProdList = new TreeSet( prodList );
        String pA, pB, aL, aR, bL, bR;

        // for all productions in the list
        alpha = newProdList.iterator();
        while( alpha.hasNext() ){

            // take one production pA
            pA = (String) alpha.next();
            // take left side of the production
            aL = pA.substring(0, pA.indexOf('-'));
            // take the right side of the production
            aR = pA.substring( pA.indexOf('-')+1, pA.length());

            // for all productions lower than pA in the list
            beta = newProdList.tailSet( pA ).iterator();
            beta.next();
            while( beta.hasNext() ){

                // update progress bar
                ProgressStatus.progressBar();

                // take one production pB
                pB = (String) beta.next();
                // take left side of the production
                bL = pB.substring(0, pB.indexOf('-'));
                // take right side of the production
                bR = pB.substring( pB.indexOf('-')+1, pB.length());

                // if the left side of pA and pB is equal
                if( aL.equals( bL ) ){

                    // remove pA from list
                    newProdList.remove( pA );
                    // remove pB from list
                    newProdList.remove( pB );
                    // compute string union of the right side of both
                    // productions aR and bR
                    aR = closure.stringUnion( aR, bR );
                    // add the new production to the list
                    newProdList.add( aL + "-" + aR );

                    // as the production list has been changed,
                    // reinitialize the list
                    alpha = newProdList.iterator();

                    // start over
                    break;
                }
            }
        }
    }
}

```

```

        }

        return newProdList;
    }

// *****
// This function removes extraneous attributes from a set of productions
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.4
private TreeSet removeExtraneous( TreeSet prodList ){

    // update progress bar text
    ProgressStatus.progressText("Removing extraneous attributes...");

    // local working variables
    TreeSet newProdList = new TreeSet( prodList );
    TreeSet prodClosure, newClosure;
    Iterator alpha;
    String prod;
    StringBuffer buff = new StringBuffer();
    int prodLen, len, mid;
    char ch;
    boolean chExtra;

    // compute closure of the given production list
    prodClosure = closure.compile( prodList );

    // for all productions in the list
    alpha = prodList.iterator();
    while( alpha.hasNext() ){
        // get one production prod
        prod = (String) alpha.next();
        // get length
        len = prod.length();

        // delete the production from list
        newProdList.remove( prod );
        // take the production in a buffer to remove char one by one
        buff.append( prod );

        // for each character in the production
        for( int i=0; i < len; i++ ){

            // update progress bar
            ProgressStatus.progressBar();
            // take one char from the production
            ch = buff.charAt(i);

            // if it is not production separating char, check if
            // it is extraneous
            if( ch != '-' ){
                // delete the char
                buff.deleteCharAt(i);
                chExtra = false;
                len = buff.length();
                mid = buff.toString().indexOf('-');

                // check if this is a valid production
                if( mid > 0 && (len-mid) > 1 ){

                    // add the production to the list
                    newProdList.add( buff.toString() );

                    // compute closure
                    newClosure = closure.compile( newProdList );

                    // check closure for equality
                    if( newClosure.equals( prodClosure ) ){
                        chExtra = true;
                    }
                }
            }
        }
    }
}

```

```

        }
        // remove the production from the list
        newProdList.remove( buff.toString() );
    }

    // ch is not extraneous, insert back
    if( chExtra == false ){
        buff.insert(i, ch);
        len++;
    }else{
        i--;
    }
}

// buff holds the productions without any extraneous attribute
newProdList.add( buff.toString() );
buff.setLength(0);
}

return newProdList;
}
}

// end of class Canonical

```

Figure 9: Canonical cover of a set of functional dependencies

```

*****  

File: Dependency.java  

LastUpdate: July 16, 2001  

Usage: Tests a relation schema decomposition for dependency preserving  

*****  

package Project.Algorithms;  

import java.util.*;  

import Project.Algorithms.Closure;  

import Project.ProgressStatus;  

*****  

// class definition  

public class Dependency{  

    // closure algorithm  

    private Closure closure;  

    // closure of the given productions  

    private TreeSet prodClosure;  

    // Fi is the set of restrictions to R and Fp is closure of Fi  

    private TreeSet Fi, Fp;  

    // dependency preservation test result  

    private boolean result;  

    // *****  

    // constructor  

    public Dependency(){  

        // instantiate closure algorithm  

        closure = new Closure();  

    }  

    // *****  

    // This function tests if the given set of relation schema decomposition  

    // is dependency preserving  

    // Reference:  

}

```

```

// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 7.3.1.2
public boolean compile( TreeSet prodList, TreeSet schemaList ){

    // start the progress bar
    ProgressStatus.startProgressBar("Compiling - Please wait");

    // local working variables
    Iterator alpha, beta;
    String Pi, Ri;
    int len;
    char ch;

    // Fi is the set of restrictions to R
    Fi = new TreeSet();

    // update progress bar text
    ProgressStatus.progressText("Computing closure...");

    // compute closure of the given set of productions
    prodClosure = closure.compile( prodList );

    // update progress bar text
    ProgressStatus.progressText("Computing closure...Done");

    // for all relations in R
    beta = schemaList.iterator();
    while( beta.hasNext() ){
        // take one relation
        Ri = (String) beta.next();

        // update progress bar text
        ProgressStatus.progressText("Testing: " + Ri);

        // for all productions in F
        alpha = prodClosure.iterator();
        while( alpha.hasNext() ){
            // take one production from F
            Pi = (String) alpha.next();

            // test if every attribute of this production is
            // present in the relation
            boolean hold = true;
            len = Pi.length();
            for( int j=0; j < len; j++ ){

                // update progress bar
                ProgressStatus.progressBar();

                ch = Pi.charAt(j);
                if( ch != '-' && Ri.indexOf(ch) == -1 ){
                    hold = false;
                    break;
                }
            }

            // add if every char in Pi is in Ri
            if( hold ){
                // Pi holds on Ri
                Fi.add( Pi );
            }
        }
    }

    // update progress bar text
    ProgressStatus.progressText("Computing closure...");

    // compute closure of Fi
    Fp = closure.compile( Fi );

    // update progress bar text

```

```

    ProgressStatus.progressText("Computing closure...Done");

    // check equality
    if( Fp.equals( prodClosure ) ){
        // dependency preserved decomposition
        result = true;
    }else{
        // decomposition is not dependency preserved
        result = false;
    }

    // update progress bar
    ProgressStatus.stopProgressBar();

    // return result
    return result;
}

// *****
// This function generates string representation of TreeSet
private String TreeSetToString( TreeSet set ){
    StringBuffer buff = new StringBuffer();

    // if set not empty
    if( set != null ){
        // for all element in set
        Iterator i = set.iterator();
        while( i.hasNext() ){
            // add each element to the string
            buff.append( (String) i.next() + "\n" );
        }
    }

    return buff.toString();
}

// *****
// This function returns the F+ computation of F during dependency test
public String getFPlus(){
    return TreeSetToString( prodClosure );
}

// *****
// This function returns the F' computation of F during dependency test
public String getFPrime(){
    return TreeSetToString( Fi );
}

// *****
// This function returns the F'+ computation of F during dependency test
public String getFPrimePlus(){
    return TreeSetToString( Fp );
}

// *****
// This function returns the result of dependency test
public String getResult(){
    if( result ){
        return "Decomposition is Dependency preserving";
    }else{
        return "Decomposition is NOT Dependency preserving";
    }
}

```

Figure 10: Dependency preservation test algorithm implementation

```

/*
File: BCNF.java
LastUpdate: November 21, 2001
Usage: Compute BCNF decomposition of relation schema
***** */

package Project.Algorithms;

import java.util.*;
import Project.Algorithms.Closure;
import Project.ProgressStatus;

// class definition ****
public class BCNF{

    // closure algorithm
    private Closure closure;

    // constructor ****
    public BCNF(){
        // instantiate new algorithm
        closure = new Closure();
    }

    // ****
    // This function decomposes a given relation R into BCNF based on
    // a set of functional dependencies
    // Reference:
    // Silberschatz, Korth, Sudarshan; Database System Concepts,
    // 3rd Edition, Section: 7.3.2
    public TreeSet compile( TreeSet R, TreeSet F ){

        // start progress bar
        ProgressStatus.startProgressBar("Compiling - Please wait");

        // result holds decomposed relations, final output
        TreeSet result = new TreeSet( R );
        // keep track of the decomposition process, true when finished

        // compute F+ of the given F
        TreeSet Fp = closure.compile( F );

        // continue till the decomposition process is not done
        while( getRelation(result, F, Fp) != null );

        // stop progress bar
        ProgressStatus.stopProgressBar();

        return result;
    }

    // ****
    // This function finds a non BCNF relation Ri in a given set of
    // relation R based on a closure set of functional dependencies F
    // Reference:
    // Silberschatz, Korth, Sudarshan; Database System Concepts,
    // 3rd Edition, Section: 7.3.2
    private String getRelation( TreeSet R, TreeSet F, TreeSet Fp ){

        // update progress
        ProgressStatus.progressText("Find relation not in BCNF...");

        // list variables
        Iterator Rlist, Flist;
        // production variables
        String Ri, Fi, alpha, beta;
        // relation bcnf or not

```

```

boolean RiBcnf;
// save the current relation schema
TreeSet oldR = new TreeSet( R );

// given R, set of relations
Rlist = R.iterator();

// for all Ri in R
while( Rlist.hasNext() ){
    // take one Ri
    Ri = (String) Rlist.next();

    // initialize, Ri is not in BCNF
    RiBcnf = false;

    // given F+, closure set of F
    Flist = Fp.iterator();

    // for all Fi in F+
    while( Flist.hasNext() ){

        // update progress bar
        ProgressStatus.progressBar();

        // take one Fi of the form alpha->beta
        Fi = (String) Flist.next();
        // take alpha
        alpha = Fi.substring(0, Fi.indexOf('-'));
        // take beta
        beta = Fi.substring(Fi.indexOf('-')+1, Fi.length());

        // if alpha and beta is subset of Ri
        if( Closure.subsetOf(Ri,alpha) != -1 &&
            Closure.subsetOf(Ri,beta) != -1 ){

            // if beta is subset of alpha
            if( Closure.subsetOf(alpha,beta) != -1 ){
                // Fi is trivial
                RiBcnf = true;
            }
            // if alpha is superkey of Ri
            else if( superkey(alpha, Ri, F) ){
                // alpha is superkey of Ri
                RiBcnf = true;
            }
            // Ri is not in BCNF
            else{
                RiBcnf = false;

                // decompose Ri into BCNF
                decompose( Ri, F, Fp, oldR );

                // if R has changed
                if( !oldR.equals( R ) ){
                    // update new R
                    R.clear();
                    R.addAll( oldR );
                    return Ri;
                }
            }
        }
    }
}

// if all the relations in R is in BCNF
return null;
}

// ****

```

```

// This function checks if a given set of attribute alpha is the
// superkey of a relation R for a given set of Functional Dependency F
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.3
public boolean superkey( String alpha, String R, TreeSet F ){

    // update progress bar text
    ProgressStatus.progressText("Testing superkey...");

    // result is the set of attributes functionally determined by alpha
    String result, oldresult;
    // list of FDs in F
    Iterator Flist;
    // FD Fi of the form beta->gama
    String Fi, beta, gama;

    // initialize result
    result = alpha;

    // compute all attributes functionally determined by alpha
    do{
        // save result for later comparison
        oldresult = result;

        // initialize Flist, list of FDs in F
        Flist = F.iterator();

        // for all FDs in F
        while( Flist.hasNext() ){

            // update progress bar
            ProgressStatus.progressBar();

            // take one FD from list, beta->gama
            Fi = (String) Flist.next();
            // take beta
            beta = Fi.substring(0, Fi.indexOf('-'));
            // take gama
            gama = Fi.substring(Fi.indexOf('-')+1, Fi.length() );

            // if beta is subset of result
            if( Closure.subsetOf(result,beta) != -1 ){
                // result = result union beta
                result = closure.stringUnion(result, gama);
            }
        }

        // if result changes, compute again
        }while( result.equals( oldresult ) == false );

        // if alpha functionally determines all attributes in R
        // then alpha is superkey of R
        if( Closure.subsetOf(result, R) != -1 ){
            // alpha is superkey of R
            return true;
        }else{
            // alpha is not the superkey of R
            return false;
        }
    }

    // ****
    // This function decomposes a relation Ri into BCNF for a given
    // set functional dependencies F
    public void decompose(String Ri, TreeSet F, TreeSet Fp, TreeSet result){

        // update progress bar text
        ProgressStatus.progressText("Decomposing relation...");
```

```

// local working variables
Iterator Flist;
String Fi, alpha, beta;
int size;

// for all Fi in F
Flist = F.iterator();
while( Flist.hasNext() ){

    // update progress bar
    ProgressStatus.progressBar();

    // take Fi from F
    Fi = (String) Flist.next();

    // Fi is non trivial and holds on Ri such that alpha -> Ri is
    // not in F+ and alpha intersection beta is null
    if( holdsOn( Ri, Fi, Fp ) ){
        // save size
        size = result.size();

        // remove Ri from the set
        result.remove( Ri );
        // take alpha of Fi = alpha -> beta
        alpha = Fi.substring( 0, Fi.indexOf('-') );
        // take beta of Fi = alpha -> beta
        beta = Fi.substring( Fi.indexOf('-')+1, Fi.length() );
        // add (alpha, beta)
        result.add( closure.stringUnion( alpha, beta ) );
        // add (Ri - beta)
        result.add( stringSubtraction( Ri, beta ) );

        // if the decomposition is successful
        if( result.size() != size ){
            break;
        }
    }
}

// *****
// This function tests if a functional dependency Fi (alpha->beta)
// holds on a relation Ri such that alpha->Ri is not in F+, and
// (alpha intersection beta) = null
private boolean holdsOn( String Ri, String Fi, TreeSet Fp ){
    String alpha, beta;

    // take alpha from Fi = alpha -> beta
    alpha = Fi.substring(0, Fi.indexOf('-'));
    // take beta from Fi = alpha -> beta
    beta = Fi.substring( Fi.indexOf('-')+1, Fi.length());

    // alpha -> beta is not trivial
    if( alpha.indexOf(beta) == -1 ){
        // alpha -> Ri is not in F+ and alpha intersection
        // beta is null
        if( !Fp.contains( alpha + "-" + Ri ) &&
            stringIntersection(alpha, beta) == true ){
            return true;
        }else{
            return false;
        }
    }else{
        return false;
    }
}

// *****
// This function checks if intersection between two relation schema

```

```

// alpha and beta is null or not
public static boolean stringIntersection( String alpha, String beta ){
    // get length
    int alphaLen = alpha.length();

    // for all char in alpha
    for( int i=0; i < alphaLen; i++ ){
        // check if the char exists in beta
        if( beta.indexOf( alpha.charAt(i) ) != -1 ){
            // alpha intersection beta is not null
            return true;
        }
    }

    // alpha intersection beta is null
    return false;
}

// *****
// This function computes string subtraction (alpha-bets)
public String stringSubtraction( String alpha, String beta ){
    // start with the alpah
    StringBuffer subtraction = new StringBuffer( alpha );
    // get length
    int len = subtraction.length();
    char ch;

    // for every char in alpha
    for( int i=0; i < len; i++ ){
        ch = subtraction.charAt(i);
        // if the char exists in beta
        if( beta.indexOf( ch ) != -1 ){
            // remove it from alpah
            subtraction.deleteCharAt(i);
            i--;
            len--;
        }
    }

    // return the intersection
    return subtraction.toString();
}
}

```

Figure 11: BCNF decomposition algorithm implementation

```

/*****
File: TNF.java
LastUpdate: November 21, 2001
Usage: Compute Third Normal Form decomposition of Relation Schema
*****/

package Project.Algorithms;

import java.util.*;
import Project.Algorithms.Closure;
import Project.Algorithms.Canonical;
import Project.ProgressStatus;

/*****
// class definition
public class TNF{
    // canonical cover algorithm
    private Canonical canonical;
    // attribute subset list

```

```

private TreeSet attrSubset;
private Object[] attrSet;
private boolean[] attr;

// ****
// constructor
public TNF(){
    // instantiate canonical cover algorithm
    canonical = new Canonical();
}

// ****
// This function decomposes a given relation R into 3NF form based on
// a set of functional dependencies
// Reference:
// Elmasri, Navathe; Fundamentals of Database System, Section: 14.1.3
public TreeSet compile( TreeSet R, TreeSet F ){

    // start progress bar
    ProgressStatus.startProgressBar("Compiling - Please wait");

    // local working variables
    String Fi, Fl, Fr, Pi, Pl, Pr, Ri, Rr;
    String Rstr, newRstr, canKey;
    Iterator alpha, beta;

    // newR holds decomposed relations, final output
    TreeSet newR = new TreeSet();

    // compute Canonical Cover of the given F
    TreeSet Fc = canonical.compile( F );

    // for each left hand side of the production that appears in Fc
    alpha = Fc.iterator();
    while( alpha.hasNext() ){
        // take the production
        Fi = (String) alpha.next();
        // take left side
        Fl = Fi.substring(0, Fi.indexOf('-'));
        // take right side
        Fr = Fi.substring( Fi.indexOf('-')+1, Fi.length());
        // union left and right side
        Ri = Closure.stringUnion( Fl, Fr );

        // for all productions from the list after Fi
        beta = Fc.tailSet( Fi ).iterator();
        while( beta.hasNext() ){
            // take one production
            Pi = (String) beta.next();
            // take left side
            Pl = Pi.substring(0, Pi.indexOf('-'));
            // take right side
            Pr = Pi.substring( Pi.indexOf('-')+1, Pi.length());

            // if the left side of the productions are equal
            if( Fl.equals( Pl ) ){
                // union right side to the result
                Ri = Closure.stringUnion( Ri, Pr );
            }
        }

        // add the new relation to list
        newR.add( Ri );
    }

    // place all remaining unplaced attributes in a single relation
    // schema
    // compute union of all given relation
}

```

```

alpha = R.iterator();
Rstr = "";
while( alpha.hasNext() ){
    Rr = (String) alpha.next();
    Rstr = Closure.stringUnion( Rstr, Rr );
}

// compute union of all decomposed relation
alpha = newR.iterator();
newRstr = "";
while( alpha.hasNext() ){
    Rr = (String) alpha.next();
    newRstr = Closure.stringUnion( newRstr, Rr );
}

// compute string subtraction of the given and decomposed
Rr = stringSubtraction( Rstr, newRstr );

// if the subtraction is not null
if( !Rr.equals("") ){
    // add the relation to the list
    newR.add( Rr );
}

// if none of the schema contains a key of R, create one more
// realtion schema that contains attributes that form a key for R
// compute all possible subsets of the attribute set
TreeSet attrList = Closure.attributes( R );
attrSet = attrList.toArray();
attrSubset = new TreeSet();
attr = new boolean[ attrSet.length + 1 ];
Subset( attrSet.length );

// get a candidate key
canKey = "";
alpha = attrSubset.iterator();
while( alpha.hasNext() ){
    Ri = (String) alpha.next();
    if( superkey( Ri, Rstr, Fc ) ){
        if( Ri.length() < canKey.length() ){
            canKey = Ri;
        }
    }
}

// if there exists one candidate key and none of the relation
// contains the candidatea key
if( !canKey.equals("") ){
    boolean addnew = true;
    alpha = newR.iterator();
    while( alpha.hasNext() ){
        Ri = (String) alpha.next();
        if( Closure.subsetOf(Ri,canKey) != -1 ){
            addnew = false;
        }
    }

    if( addnew ){
        // add the key to as a new relation
        newR.add( canKey );
    }
}

// stop the progress bar
ProgressStatus.stopProgressBar();

// new relation schema
return newR;
}

```

```

// *****
// This function tests if intersection between two relation is null
public static boolean stringIntersection( String alpha, String beta ){
    // get length of alpha
    int alphalen = alpha.length();

    // for all chars in alpha
    for( int i=0; i < alphalen; i++ ){
        // test if the char is present in beta
        if( beta.indexOf( alpha.charAt(i) ) != -1 ){
            // if present then intersection is not null
            return false;
        }
    }

    // intersection is null
    return true;
}

// *****
// This function computes set subtraction on two string parameters
public String stringSubtraction( String alpha, String beta ){
    // start with the full string
    StringBuffer subtraction = new StringBuffer( alpha );
    // get length
    int len = subtraction.length();
    char ch;

    // for each char in alpha
    for( int i=0; i < len; i++ ){
        // take the char
        ch = subtraction.charAt(i);
        // if it is present in beta
        if( beta.indexOf( ch ) != -1 ){
            // delete the char from alpha
            subtraction.deleteCharAt(i);
            // update loop variables
            i--;
            len--;
        }
    }

    // return subtraction
    return subtraction.toString();
}

// *****
// This function generates all possible subsets of the attribute list
private void Subset( int n ){
    if( n == 0 ){
        writeSubset();
    }else{
        attr[ n ] = false; Subset( n-1 );
        attr[ n ] = true; Subset( n-1 );
    }
}

// *****
// This function writes subset computed by Subset function
private void writeSubset(){
    StringBuffer strBuff = new StringBuffer();

    try{

```

```

// compute the subset
for( int i=1; i <= attrSet.length; i++ ){
    if( attr[i] == true ){
        strBuff.append( attrSet[i-1] );
    }
}

// add the subset to the subset list
if( strBuff.length() > 0 ){
    attrSubset.add( strBuff.toString() );
}
}catch( Exception err ){
    System.out.println("writeSubset: " + err );
}

}

// *****
// This function checks if a given set of attribute alpha is the
// superkey of a relation R for a given set of Functional Dependency F
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.3
public boolean superkey( String alpha, String R, TreeSet F ){

    // update progress bar text
    ProgressStatus.progressText("Testing superkey...");

    // result is the set of attributes functionally
    // determined by alpha
    String result, oldresult;
    // list of FDs in F
    Iterator Flist;
    // FD Fi of the form beta->gama
    String Fi, beta, gama;

    // initialize result
    result = alpha;

    // compute all attributes functionally determined by alpha
    do{
        // save result for later comparison
        oldresult = result;

        // initialize Flist, list of FDs in F
        Flist = F.iterator();

        // for all FDs in F
        while( Flist.hasNext() ){

            // update progress bar
            ProgressStatus.progressBar();

            // take one FD from list, beta->gama
            Fi = (String) Flist.next();
            // take beta
            beta = Fi.substring(0, Fi.indexOf('-'));
            // take gama
            gama = Fi.substring(Fi.indexOf('-')+1, Fi.length() );

            // if beta is subset of result
            if( result.indexOf(beta) != -1 ){
                // result = result union beta
                result = Closure.stringUnion(result, gama);
            }
        }

        // if result changes, compute again
       }while( result.equals( oldresult ) == false );
    }
}

```

```

        // if alpha functionally determines all attributes in R
        // then alpha is superkey of R
        if( Closure.subset(result, R) != -1 ){
            // alpha is superkey of R
            return true;
        }else{
            // alpha is not superkey of R
            return false;
        }
    }
}

```

Figure 12: 3NF decomposition algorithm implementation

```

*****
File: BasicProject.java
LastUpdate: November 21, 2001
Usage: Provides basic structure of the database design algorithm interface
*****


package Project;

import javax.swing.JFrame;
import javax.swing.JMenuBar;
import javax.swing.JTextArea;
import javax.swing.JOptionPane;
import javax.swing.JFileChooser;

import java.io.*;
import java.util.*;

// ****
// class definition
public abstract class BasicProject{

    // dialog box option button text
    private static final String [] label1 = { "Yes", "No" };
    private static final String [] label2 = { "Yes", "No", "Cancel" };

    // application window
    protected static JFrame window;

    // application menu bar
    protected static JMenuBar menuBar;

    // application output window
    protected static JTextArea output;

    // file chooser dialog box
    private static JFileChooser fc;

    protected ProgressStatus ps;

    // tracks whether recompilation needed or not
    protected boolean recompile;

    // tracks whether project is saved or not
    protected boolean saveProject;

    // file to be saved in
    private File file;

    // ****
    // constructor, initialization
}

```

```

public BasicProject( JFrame mainWindow, JMenuBar mainMenuBar,
JTextArea mainWindowOutput ){

    // initialize variables
    window = mainWindow;
    menuBar = mainMenuBar;
    output = mainWindowOutput;

    // initialize built-in file chooser window
    fc = new JFileChooser();
    // initialize progress bar
    ps = new ProgressStatus( window, "Progress Bar" );

    // control variables
    recompile = true;
    saveProject = true;
    file = null;
}

// *****
// save current project
public void save(){

    // if there is a selected file
    if( file != null ){
        try{
            // open a new file writer
            FileWriter fw = new FileWriter( file );
            // write the output window contents to the file
            fw.write( output.getText() );
            // close file
            fw.close();
            // reset control variables
            saveProject = false;
        }catch(Exception e){
            // only if error occurs during file saving
            JOptionPane.showMessageDialog(window,
                "Error saving project\n" + e.toString());
            file = null;
        }
        // return as saving is already done
        return;
    }

    boolean retry = false;

    do{
        // popup file save dialog box
        int opt = fc.showSaveDialog(window);

        // if user press ok button
        if( opt == fc.APPROVE_OPTION ){

            // get the selected file
            file = fc.getSelectedFile();

            // file already exists
            if( file.exists() ){

                // popup overwriter permission window
                int fopt = JOptionPane.showOptionDialog(window,
                    "File already exists.\nDo you want to overwrite?",
                    "Confirm overwrite",
                    JOptionPane.YES_NO_CANCEL_OPTION,
                    JOptionPane.QUESTION_MESSAGE, null, label2,
                    label2[1]);

                // overwrite file
                if( fopt == JOptionPane.YES_OPTION ){
                    try{

```

```

        }
    }

    // do not overwrite file
    else if( fopt == JOptionPane.NO_OPTION ){
        // retry
        file = null;
        retry = true;
    }
    else{
        // cancel save
        file = null;
        retry = false;
    }
}

// save
else{
    try{
        FileWriter fw = new FileWriter(file);
        fw.write( output.getText() );
        fw.close();
        saveProject = false;
    }catch( Exception e ){
        JOptionPane.showMessageDialog(window,
            "Error saving project\n" + e.toString());
        file = null;
    }
}
else{
    retry = false;
}
}

while( retry == true );
}

// *****
// print to printer
public void print(){
    // left for later implementation
    // System.out.println("Printing...");
}

// *****
// close current project
public void close(){
    // check if project needs to be saved
    if( saveProject == true ){

        // popup save option window
        int fopt = JOptionPane.showOptionDialog(window,
            "Project not saved.\nDo you want to save it now?",
            "Save Project", JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE, null, label1, label1[0]);

        if( fopt == JOptionPane.YES_OPTION ){
            save();
        }
    }
}

// remove menu options
menuBar.remove(1);

```

```
// clear output window
output.setText("");
}

// *****
// converts a TreeSet to string
public static String TreeSetToString( TreeSet set ){
    StringBuffer buff = new StringBuffer();

    // if set is not null
    if( set != null ){
        // loop through the set
        Iterator i = set.iterator();
        while( i.hasNext() ){
            // append every element to the string
            buff.append( (String) i.next() + "\n" );
        }
    }

    // return the string representation
    return buff.toString();
}
```

Figure 13: Basic project structure implementation.

```

*****  

File: AClosure.java  

LastUpdate: July 16, 2001  

Usage: Provides an user interface to the Closure algorithm  

*****  

package Project;  

import java.awt.*;  

import java.awt.event.*;  

import java.util.TreeSet;  

import java.util.Iterator;  

import javax.swing.JFrame;  

import javax.swing.JMenu;  

import javax.swing.JMenuBar;  

import javax.swing.JMenuItem;  

import javax.swing.JTextArea;  

import Project.Algorithms.Closure;  

import Project.BasicProject;  

import Project.ProgressStatus;  

*****  

// class definition  

public class AClosure extends BasicProject implements ActionListener{  

    // input space  

    private TreeSet f, r;  

    // output space  

    private String ap;  

    // input form variable  

    private InputForm iff, ifr;  

    // *****  

    // constructor  

    public AClosure(JFrame mainWindow, JMenuBar mainMenuBar,  

        JTextArea mainWindowOutput ){  

        // initialize super class  

        super(mainWindow, mainMenuBar, mainWindowOutput);  

        // initialize menu  

        JMenu menu = new JMenu("Closure of Attribute Sets");  

        menuBar.add(menu);  

        // add "Input: Alpha - Set of Attributes" menu item  

        JMenuItem menuItem = new  

            JMenuItem("Input: Alpha - Set of Attributes");  

        menuItem.addActionListener(this);  

        menu.add(menuItem);  

        // add "Input: F - Set of Functional Dependencies" menu item  

        menuItem = new  

            JMenuItem("Input: F - Set of Functional Dependencies");  

        menuItem.addActionListener(this);  

        menu.add(menuItem);  

        // add menu seperator  

        menu.addSeparator();  

        // add "Compile" menu item  

        menuItem = new JMenuItem("Compile");  

        menuItem.addActionListener(this);  

        menuItem.setEnabled(false);  

        menu.add(menuItem);  

    }  

}

```

```

        // initialize Relation Schema input windows
        ifr = new InputForm( window, "Input: Alpha - Set of Attributes", 2);

        // initialize Functional Dependency input windows
        iff = new InputForm( window,
                            "Input: F - Set of Functional Dependencies", 1);

        // initialize input space
        f = new TreeSet();
        r = new TreeSet();

        // update output window
        output();
    }

    // *****
    // menu event action handler
    public void actionPerformed(ActionEvent e) {
        // get event source menu item
        JMenuItem source = (JMenuItem)(e.getSource());

        // get event source menu item text
        String action = source.getText();

        // perform action based on the selected menu option
        if( action.equals("Input: Alpha - Set of Attributes") ){
            // relation schema input window
            rInputWindow();
        }
        else if( action.equals("Input: F - Set of Functional Dependencies") ){
            // functional dependency input window
            fInputWindow();
        }
        else if( action.equals("Compile") ){
            // convert relation to one single string
            StringBuffer alpha = new StringBuffer();
            Iterator i = r.iterator();
            while( i.hasNext() ){
                alpha.append( (Character)i.next() );
            }
            // compute algorithm
            ap = compile( alpha.toString(), f );
            // compile not needed
            recompile = false;
            // not saved
            saveProject = true;
            // update output window
            output();
        }
    }

    // *****
    // relation schema input form window
    private void rInputWindow(){
        // variable to keep track whether the input
        // has changed since last compile
        boolean changed;

        // take input from user
        changed = ifr.showDialog( window, r );

        // check if input has changed
        if( changed ){
            // recompile needed
            recompile = true;
            // not saved
            saveProject = true;
        }
    }
}

```

```

        // update output window
        output();
    }

}

// *****
// functional dependency input form window
private void fInputWindow() {
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = iff.showDialog( window, f );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

    // enable/disable compile menu option
    if( f.isEmpty() ){
        menuBar.getMenu(1).getItem(3).setEnabled(false);
    }else{
        menuBar.getMenu(1).getItem(3).setEnabled(true);
    }
}

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

    // print project title
    sb.append("[Project]\n");
    sb.append("Closure of Attribute Sets\n\n");

    // print input R
    if( !r.isEmpty() ){
        sb.append("[Input: Alpha - Set of Attributes]\n");
        Iterator i = r.iterator();
        while( i.hasNext() ){
            sb.append( (Character) i.next() + "\n" );
        }
        sb.append( "\n" );
    }

    // print input F
    if( !f.isEmpty() ){
        sb.append("[Input: F - Set of " +
                  "Functional Dependencies]\n");
        sb.append( TreeSetToString( f ) + "\n" );
    }

    // print output
    if( recompile == false ){
        sb.append("[Output: Closure of Attribute Sets]\n");
        sb.append( ap + "\n" );
    }

    // print ot application output window
    output.setText( sb.toString() );
}

```

```

// ****
// This function finds all attributes functionally determined by a
// given set of attributes and functional dependency
// Reference:
// Silberschatz, Korth, Sudarshan; Database System Concepts,
// 3rd Edition, Section: 6.5.3
private String compile( String alpha, TreeSet F ){

    // start progress bar
    ProgressStatus.startProgressBar("Compiling - Please wait");

    // result is the set of attributes functionally
    // determined by alpha
    String result, oldresult;
    // list of FDs in F
    Iterator Flist;
    // FD Fi of the form beta->gama
    String Fi, beta, gama;

    // initialize result
    result = alpha;

    // compute all attributes functionally determined by alpha
    do{
        // save result for later comparison
        oldresult = result;

        // initialize Flist, list of FDs in F
        Flist = F.iterator();

        // for all FDs in F
        while( Flist.hasNext() ){
            // take one FD from list, beta->gama
            Fi = (String) Flist.next();
            // take beta
            beta = Fi.substring(0, Fi.indexOf('-'));
            // take gama
            gama = Fi.substring(Fi.indexOf('-')+1, Fi.length() );

            // if beta is subset of result
            if( Closure.subsetOf(result,beta) != -1 ){
                // result = result union beta
                result = Closure.stringUnion(result, gama);
            }
        }

        ProgressStatus.progressBar();
    }

    // if result changes, compute again
}while( result.equals( oldresult ) == false );

    // end progress bar
    ProgressStatus.stopProgressBar();

    return result;
}
}

```

Figure 14: User interface to the Closure algorithm

```

/*****
File: CCover.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to the canonical cover algorithm
*****/
package Project;

```

```

import java.awt.*;
import java.awt.event.*;

import java.util.TreeSet;
import java.util.Iterator;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;

import Project.Algorithms.Canonical;
import Project.BasicProject;

// *****
// class definition
public class CCover extends BasicProject implements ActionListener{

    // input space
    private TreeSet f;

    // output space
    private TreeSet fc;

    // input form variable
    private InputForm iff;

    // algorithm
    private Canonical canonical;

    // *****
    // constructor, initialization
    public CCover(JFrame mainWindow,
                 JMenuBar mainMenuBar, JTextArea mainWindowOutput ){
        // initialize super class
        super(mainWindow, mainMenuBar, mainWindowOutput);

        // add "Canonical Cover" project menu to the menu bar
        JMenu menu = new JMenu("Canonical Cover");
        menuBar.add(menu);

        // add "Input: F - Set of Functional Dependencies" menu item
        JMenuItem menuItem = new
                           JMenuItem("Input: F - Set of Functional Dependencies");
        menuItem.addActionListener(this);
        menu.add(menuItem);

        menu.addSeparator();

        // add "Compile" menu item
        menuItem = new JMenuItem("Compile");
        menuItem.addActionListener(this);
        menuItem.setEnabled(false);
        menu.add(menuItem);

        // initialize Functional Dependency input windows
        iff = new InputForm( window,
                           "Input: F - Set of Functional Dependencies", 1);

        // initialize input space
        f = new TreeSet();

        // initialize algorithm
        canonical = new Canonical();

        // update output window
        output();
    }
}

```

```

}

// *****
// menu event action handler
public void actionPerformed(ActionEvent e) {
    // get event source menu item
    JMenuItem source = (JMenuItem)(e.getSource());

    // get event source menu item text
    String action = source.getText();

    if( action.equals("Input: F - Set of Functional Dependencies") ){
        // functional dependency input window
        fInputWindow();
    }
    else if( action.equals("Compile") ){
        fc = canonical.compile( f );
        // compile not needed
        recompile = false;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

// *****
// functional dependency input form window
private void fInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = iff.showDialog( window, f );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

    // enable/disable compile menu option
    if( f.isEmpty() ){
        menuBar.getMenu(1).getItem(2).setEnabled(false);
    }else{
        menuBar.getMenu(1).getItem(2).setEnabled(true);
    }
}

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

    // print project title
    sb.append("[Project]\n");
    sb.append("Canonical Cover\n\n");

    // print input F
    if( !f.isEmpty() ){
        sb.append("[Input: F - Set of Functional Dependencies]\n");
        sb.append( TreeSetToString( f ) + "\n" );
    }
}

```

```

        // print output
        if( recompile == false ){
            sb.append("[Output: Fc - Canonical Cover of F]\n");
            sb.append( TreeSetToString( fc ) + "\n" );
        }

        // print ot application output window
        output.setText( sb.toString() );
    }
}

```

Figure 15: User interface to the Canonical Cover algorithm

```

/*
File: DTest.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to the Dependency Preservation Test
***** */

package Project;

import java.awt.*;
import java.awt.event.*;

import java.util.TreeSet;
import java.util.Iterator;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;

import Project.Algorithms.Dependency;
import Project.BasicProject;

// ****
// class definition
public class DTest extends BasicProject implements ActionListener{

    // input space
    private TreeSet f, r;

    // output space
    private boolean result;

    // input form variable
    private InputForm iff, ifr;

    // algorithm
    private Dependency dependency;

    // ****
    // constructor, initialization
    public DTest(JFrame mainWindow,
                JMenuBar mainMenuBar, JTextArea mainWindowOutput ){
        super(mainWindow, mainMenuBar, mainWindowOutput);

        // add "Dependency Preservation Test" project
        // menu to the menu bar
        JMenu menu = new JMenu("Dependency Preservation Test");
        mainMenuBar.add(menu);

        // add "Input: R - Relation Schema" menu item
        JMenuItem menuItem = new JMenuItem("Input: R - Relation Schema");

```

```

menuItem.addActionListener(this);
menu.add(menuItem);

// add "Input: F - Set of Functional Dependencies" menu item
menuItem = new JMenuItem("Input: F - Set of Functional Dependencies");
menuItem.addActionListener(this);
menu.add(menuItem);

menu.addSeparator();

// add "Compile" menu item
menuItem = new JMenuItem("Compile");
menuItem.addActionListener(this);
menuItem.setEnabled(false);
menu.add(menuItem);

// initialize Relation Schema input windows
ifr = new InputForm( window, "Input: R - Relation Schema", 3);

// initialize Functional Dependency input windows
iff = new InputForm( window,
                    "Input: F - Set of Functional Dependencies", 1);

// initialize input space
f = new TreeSet();
r = new TreeSet();

// initialize algorithm
dependency = new Dependency();

// update output window
output();
}

// *****
// menu event action handler
public void actionPerformed(ActionEvent e) {
    // get event source menu item
    JMenuItem source = (JMenuItem)(e.getSource());

    // get event source menu item text
    String action = source.getText();

    // perform action based on the selected menu option
    if( action.equals("Input: R - Relation Schema") ){
        // relation schema input window
        rInputWindow();
    }
    else if(
        action.equals("Input: F - Set of Functional Dependencies")){
        // functional dependency input window
        fInputWindow();
    }
    else if( action.equals("Compile") ){
        result = dependency.compile( f, r );
        // compile not needed
        recompile = false;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

// *****
// relation schema input form window
private void rInputWindow(){
    // variable to keep track whether the input
}

```

```

// has changed since last compile
boolean changed;

// take input from user
changed = ifr.showDialog( window, r );

// check if input has changed
if( changed ){
    // recompile needed
    recompile = true;
    // not saved
    saveProject = true;
    // update output window
    output();
}

// *****
// functional dependency input form window
private void fInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = iff.showDialog( window, f );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

    // enable/disable compile menu option
    if( f.isEmpty() ){
        menuBar.getMenu(1).getItem(3).setEnabled(false);
    }else{
        menuBar.getMenu(1).getItem(3).setEnabled(true);
    }
}

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

    // print project title
    sb.append("[Project]\n");
    sb.append("Dependency Preservation Test\n\n");

    // print input R
    if( !r.isEmpty() ){
        sb.append("[Input: R - Relation Schema]\n");
        sb.append( TreeSetToString( r ) + "\n" );
    }

    // print input F
    if( !f.isEmpty() ){
        sb.append("[Input: F - Set of Functional Dependencies]\n");
        sb.append( TreeSetToString( f ) + "\n" );
    }

    // print output
    if( recompile == false ){
        sb.append("[F+ Closure of F]\n");
    }
}

```

```

        sb.append( dependency.getFPlus() + "\n" );
        sb.append(" [F' Functional Dependencies on schema R]\n");
        sb.append( dependency.getFPrime() + "\n" );

        sb.append(" [F'+ Closure of F']\n";
        sb.append( dependency.getFPrimePlus() + "\n" );

        sb.append(" [Output: Dependency Preservation Test]\n");
        sb.append( dependency.getResult() + "\n" );
    }

    // print ot application output window
    output.setText( sb.toString() );
}
}

```

Figure 16: User interface to the Dependency Preservation Test

```

/*
File: FDClosure.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to the Closure algorithm
***** */

package Project;

import java.awt.*;
import java.awt.event.*;

import java.util.TreeSet;
import java.util.Iterator;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;

import Project.Algorithms.Closure;
import Project.BasicProject;

// *****
// class definition
public class FDClosure extends BasicProject implements ActionListener{

    // input space
    private TreeSet f, r;

    // output space
    private TreeSet fp;

    // input form variable
    private InputForm iff, ifr;

    // algorithm
    private Closure closure;

    // *****
    // constructor, initialization
    public FDClosure(JFrame mainWindow,
                     JMenuBar mainMenuBar, JTextArea mainWindowOutput ){

        // initialize constructor
        super(mainWindow, mainMenuBar, mainWindowOutput);

        // add "Closure of a Set of Functional Dependencies" project
}

```

```

// menu to the menu bar
JMenu menu = new
    JMenu("Closure of a Set of Functional Dependencies");
menuBar.add(menu);

// add "Input: R - Relation Schema" menu item
JMenuItem menuItem = new JMenuItem("Input: R - Relation Schema");
menuItem.addActionListener(this);
menu.add(menuItem);

// add "Input: F - Set of Functional Dependencies" menu item
menuItem = new
    JMenuItem("Input: F - Set of Functional Dependencies");
menuItem.addActionListener(this);
menu.add(menuItem);

menu.addSeparator();

// add "Compile" menu item
menuItem = new JMenuItem("Compile");
menuItem.addActionListener(this);
menuItem.setEnabled(false);
menu.add(menuItem);

// initialize Relation Schema input windows
ifr = new InputForm( window, "Input: R - Relation Schema", 3);

// initialize Functional Dependency input windows
iff = new InputForm( window,
    "Input: F - Set of Functional Dependencies", 1);

// initialize input space
f = new TreeSet();
r = new TreeSet();

// initialize algorithm
closure = new Closure();

// update output window
output();
}

// *****
// menu event action handler
public void actionPerformed(ActionEvent e) {
    // get event source menu item
    JMenuItem source = (JMenuItem) (e.getSource());

    // get event source menu item text
    String action = source.getText();

    // perform action based on the selected menu option
    if( action.equals("Input: R - Relation Schema") ){
        // relation schema input window
        rInputWindow();
    }
    else if(
        action.equals("Input: F - Set of Functional Dependencies") ){
        // functional dependency input window
        fInputWindow();
    }
    else if( action.equals("Compile") ){
        fp = closure.compile( f, r );
        // compile not needed
        recompile = false;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

```

```

}

// *****
// relation schema input form window
private void rInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = ifr.showDialog( window, r );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

// *****
// functional dependency input form window
private void fInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = iff.showDialog( window, f );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

    // enable/disable compile menu option
    if( f.isEmpty() ){
        menuBar.getMenu(1).getItem(3).setEnabled(false);
    }else{
        menuBar.getMenu(1).getItem(3).setEnabled(true);
    }
}

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

    // print project title
    sb.append("[Project]\n");
    sb.append("Closure of a Set of Functional Dependencies\n\n");

    // print input R
    if( !r.isEmpty() ){
        sb.append("[Input: R - Relation Schema]\n");
        sb.append( TreeSetToString( r ) + "\n" );
    }

    // print input F
    if( !f.isEmpty() ){

```

```

        sb.append("[Input: F - Set of Functional Dependencies]\n");
        sb.append( TreeSetToString( f ) + "\n" );
    }

    // print output
    if( recompile == false ){
        sb.append("[Attribute List]\n");
        sb.append( closure.getAttributes() + "\n" );

        sb.append("[Attribute Subset]\n");
        sb.append( closure.getAttributeSubset() + "\n" );

        sb.append("[Output: Reflexivity]\n");
        sb.append( closure.getReflexivity() + "\n" );

        sb.append("[Output: Augmentation]\n");
        sb.append( closure.getAugmentation() + "\n" );

        sb.append("[Output: Transitivity]\n");
        sb.append( closure.getTransitivity() + "\n" );

        sb.append("[Output: Closure Set of Functional Dependencies]\n");
        sb.append( TreeSetToString( fp ) + "\n" );
    }

    // print ot application output window
    output.setText( sb.toString() );
}
}

```

Figure 17: User interface to the Closer of Functional Dependency algorithm

```

*****
File: BCNFcom.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to the BCNF decomposition algorithm
***** 

package Project;

import java.awt.*;
import java.awt.event.*;

import java.util.TreeSet;
import java.util.Iterator;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;

import Project.Algorithms.BCNF;
import Project.BasicProject;

// *****
// class definition
public class BCNFcom extends BasicProject implements ActionListener{

    // input space
    private TreeSet f, r;

    // output space
    private TreeSet fp;

    // input form variable
    private InputForm iff, ifr;
}

```

```

// algorithm
private BCNF bcnf;

// ****
// constructor, initialization
public BCNFDcom(JFrame mainWindow,
                JMenuBar mainMenuBar, JTextArea mainWindowOutput ) {

    // initialize constructor
    super(mainWindow, mainMenuBar, mainWindowOutput);

    // menu to the menu bar
    JMenu menu = new JMenu("BCNF Decomposition");
    menuBar.add(menu);

    // add "Input: R - Relation Schema" menu item
    JMenuItem menuItem = new JMenuItem("Input: R - Relation Schema");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // add "Input: F - Set of Functional Dependencies" menu item
    menuItem = new JMenuItem("Input: F - Set of Functional Dependencies");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    menu.addSeparator();

    // add "Compile" menu item
    menuItem = new JMenuItem("Compile");
    menuItem.addActionListener(this);
    menuItem.setEnabled(false);
    menu.add(menuItem);

    // initialize Relation Schema input windows
    ifr = new InputForm( window, "Input: R - Relation Schema", 3 );

    // initialize Functional Dependency input windows
    iff = new InputForm( window,
                        "Input: F - Set of Functional Dependencies", 1 );

    // initialize input space
    f = new TreeSet();
    r = new TreeSet();

    // initialize algorithm
    bcnf = new BCNF();

    // update output window
    output();
}

// ****
// menu event action handler
public void actionPerformed(ActionEvent e) {
    // get event source menu item
    JMenuItem source = (JMenuItem)(e.getSource());

    // get event source menu item text
    String action = source.getText();

    // perform action based on the selected menu option
    if( action.equals("Input: R - Relation Schema") ){
        // relation schema input window
        rInputWindow();
    }
    else if(
        action.equals("Input: F - Set of Functional Dependencies") ){

```

```

        // functional dependency input window
        fInputWindow();
    }
    else if( action.equals("Compile") ){
        fp = bcnf.compile( r, f );
        // compile not needed
        recompile = false;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

// *****
// relation schema input form window
private void rInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = ifr.showDialog( window, r );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

// *****
// functional dependency input form window
private void fInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = iff.showDialog( window, f );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

    // enable/disable compile menu option
    if( f.isEmpty() ){
        menuBar.getMenu(1).getItem(3).setEnabled(false);
    }else{
        menuBar.getMenu(1).getItem(3).setEnabled(true);
    }
}

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

```

```

        // print project title
        sb.append("[Project]\n");
        sb.append("BCNF Decomposition\n\n");

        // print input R
        if( !r.isEmpty() ){
            sb.append("[Input: R - Relation Schema]\n");
            sb.append( TreeSetToString( r ) + "\n" );
        }

        // print input F
        if( !f.isEmpty() ){
            sb.append("[Input: F - Set of Functional Dependencies]\n");
            sb.append( TreeSetToString( f ) + "\n" );
        }

        // print output
        if( recompile == false ){
            sb.append("[Output: BCNF Decomposition of R]\n");
            sb.append( TreeSetToString( fp ) + "\n" );
        }

        // print ot application output window
        output.setText( sb.toString() );
    }
}

```

Figure 18: User interface to the BCNF decomposition algorithm

```

/*
File: AutoDesign.java
LastUpdate: November 21, 2001
Usage: Provides an user interface to the automated database design algorithm
*/

package Project;

import java.awt.*;
import java.awt.event.*;

import java.util.TreeSet;
import java.util.Iterator;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;

import Project.Algorithms.BCNF;
import Project.Algorithms.TNF;
import Project.Algorithms.Dependency;
import Project.BasicProject;

// ****
// class definition
public class AutoDesign extends BasicProject implements ActionListener{

    // input space
    private TreeSet f, r;

    // output space
    private TreeSet rb, rt;
    private boolean dp;

    // input form variable
    private InputForm iff, ifr;
}

```

```

// algorithm
private BCNF bcnf;
private TNF tnf;
private Dependency dtest;

// ****
// constructor, initialization
public AutoDesign(JFrame mainWindow,
                  JMenuBar mainMenuBar, JTextArea mainWindowOutput ) {
    super(mainWindow, mainMenuBar, mainWindowOutput);

    // initialize constructor
    // menu to the menu bar
    JMenu menu = new JMenu("Automated Design");
    menuBar.add(menu);

    // add "Input: R - Relation Schema" menu item
    JMenuItem menuItem = new JMenuItem("Input: R - Relation Schema");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // add "Input: F - Set of Functional Dependencies" menu item
    menuItem = new JMenuItem("Input: F - Set of Functional Dependencies");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    menu.addSeparator();

    // add "Compile" menu item
    menuItem = new JMenuItem("Compile");
    menuItem.addActionListener(this);
    menuItem.setEnabled(false);
    menu.add(menuItem);

    // initialize Relation Schema input windows
    ifr = new InputForm( window, "Input: R - Relation Schema", 3 );

    // initialize Functional Dependency input windows
    iff = new InputForm( window,
                        "Input: F - Set of Functional Dependencies", 1 );

    // initialize input space
    f = new TreeSet();
    r = new TreeSet();

    // initialize algorithm
    bcnf = new BCNF();
    tnf = new TNF();
    dtest = new Dependency();

    // update output window
    output();
}

// ****
// menu event action handler
public void actionPerformed(ActionEvent e) {
    // get event source menu item
    JMenuItem source = (JMenuItem)(e.getSource());

    // get event source menu item text
    String action = source.getText();

    // perform action based on the selected menu option
    if( action.equals("Input: R - Relation Schema") ){
        // relation schema input window

```

```

        }

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

    // print project title
    sb.append("[Project]\n");
    sb.append("Automated Design\n\n");

    // print input R
    if( !r.isEmpty() ){
        sb.append("[Input: R - Relation Schema]\n");
        sb.append( TreeSetToString( r ) + "\n" );
    }

    // print input F
    if( !f.isEmpty() ){
        sb.append("[Input: F - Set of Functional Dependencies]\n");
        sb.append( TreeSetToString( f ) + "\n" );
    }

    // print output
    if( recompile == false ){
        if( dp ){
            sb.append("[Output: BCNF Decomposition of R]\n");
            sb.append( TreeSetToString( rb ) + "\n" );
        }else{
            sb.append("[Output: 3NF Decomposition of R]\n");
            sb.append( TreeSetToString( rt ) + "\n" );
        }
    }

    // print ot application output window
    output.setText( sb.toString() );
}
}

```

Figure 19: User interface to the semi-automated design decision algorithm

```

/*
File: InputForm.java
LastUpdate: July 16, 2001
Usage: Provides a generic user interface for user input
******/

package Project;

import java.awt.*;
import java.awt.event.*;

import javax.swing.JDialog;
import javax.swing.JList;
import javax.swing.ListSelectionModel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JOptionPane;

import java.util.*;

```

```

// ****
// class definition
public class InputForm extends JDialog {

    // initial list and the current list that user uses to modify
    private TreeSet currentlist, initiallist;
    // input form type
    private int formtype;
    // input data list
    private JList list;
    // parent frame
    private Frame frame;
    // input data has changed since load
    private boolean changed;

    // ****
    // constructor, initialization
    public InputForm(Frame frame, String title, int type) {
        // initialize super class
        super(frame, title, true);

        // initialize form type type
        formtype = type;
        // initialize current list
        currentlist = new TreeSet();

        // user input text field
        final JTextField textField = new JTextField(30);

        // add button
        final JButton addButton = new JButton("Add");
        // add button event handler
        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String item = textField.getText();
                if( item != null && item.length() > 0 && addCheck(item) ) {
                    if( formtype == 1 ){
                        String lf = item.substring(0, item.indexOf('-'));
                        String rt = item.substring( item.indexOf('-')+1,
                            item.length());
                        currentlist.add( inOrder(lf) + "-" + inOrder(rt) );
                    }else if( formtype == 2 ){
                        currentlist.add( new Character(item.charAt(0)) );
                    }else if( formtype == 3 ){
                        currentlist.add( inOrder(item) );
                    }
                    list.setListData(currentlist.toArray());
                    list.setSelectedValue(item, true);
                }
            }
        });
        // add text field event handler
        textField.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addButton.doClick();
            }
        });
        // remove button
        final JButton removeButton = new JButton("Remove");
        // add remove button event handler
        removeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if( formtype == 1 || formtype == 3){
                    String item = (String) list.getSelectedValue();
                    currentlist.remove( item );
                    list.setListData(currentlist.toArray());
                }
            }
        });
    }
}

```

```

        }else if( formtype == 2 ){
            Character item = (Character) list.getSelectedValue();
            currentlist.remove( item );
            list.setListData(currentlist.toArray());
        }
    });

    // ok button
    JButton okButton = new JButton("Ok");
    // add ok button event handler
    okButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if( currentlist.equals( initialist ) ){
                changed = false;
            }else{
                initialist.clear();
                initialist.addAll(currentlist);
                currentlist.clear();
                changed = true;
            }
            setVisible(false);
        }
    });
    // cancel button
    JButton cancelButton = new JButton("Cancel");
    // add cancel button event handler
    cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            currentlist.clear();
            changed = false;
            setVisible(false);
        }
    });
    // set default button for the window
    getRootPane().setDefaultButton( cancelButton );

    // initialize list window
    list = new JList();
    list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
    // set double click event handler
    list.addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            if (e.getClickCount() == 2) {
                removeButton.doClick();
            }
        }
    });
    // add scroll bar to the list
    JScrollPane listScroller = new JScrollPane(list);
    listScroller.setAlignmentX(LEFT_ALIGNMENT);

    //Create a container
    JPanel topPane = new JPanel();
    topPane.add(textField);
    topPane.add(addButton);
    topPane.add(removeButton);

    // set bottom panel
    JPanel bottomPane = new JPanel();
    bottomPane.add(okButton);
    bottomPane.add(cancelButton);

    //Put everything together, using the content pane's BorderLayout.
    Container contentPane = getContentPane();
    contentPane.add(topPane, BorderLayout.NORTH);
    contentPane.add(listScroller, BorderLayout.CENTER);
    contentPane.add(bottomPane, BorderLayout.SOUTH);

```

```

        pack();
    }

    // *****
    // Show the initialized dialog. The first argument should
    // be null if you want the dialog to come up in the center
    // of the screen. Otherwise, the argument should be the
    // component on top of which the dialog should appear.
    public boolean showDialog(Component comp, TreeSet listdata) {
        // save the initial list
        initiallist = listdata;
        // temporary space
        currentlist.addAll(listdata);

        list.setListData( currentlist.toArray() );
        setLocationRelativeTo(comp);
        setVisible(true);

        return changed;
    }

    // *****
    // input data check before adding to the list
    private boolean addCheck(String data){
        // FD = {F1, F2, F3} input form
        if( formtype == 1 ){
            // production separator char '-' check
            if( data.indexOf('-') == -1 ){
                JOptionPane.showMessageDialog(frame,
                    "Functional dependency must contain one separating character.\n"+
                    "Example: A-BC",
                    "Functional dependency input error",
                    JOptionPane.ERROR_MESSAGE);
                return false;
            }
            // length of the left side of the production
            else if( data.substring(0, data.indexOf('-')).length() == 0 ){
                JOptionPane.showMessageDialog(frame,
                    "Not a valid functional dependency.",
                    "Functional dependency input error",
                    JOptionPane.ERROR_MESSAGE);
                return false;
            }
            // length of the right side of the production
            else if( data.substring(data.indexOf('-')+1,
                data.length()).length() == 0 ){
                JOptionPane.showMessageDialog(frame,
                    "Not a valid functional dependency.",
                    "Functional dependency input error",
                    JOptionPane.ERROR_MESSAGE);
                return false;
            }
            else if( !validCharCheck(data) ){
                JOptionPane.showMessageDialog(frame,
                    "Functional dependency must be composed of:\n" +
                    "letters, digits and one separating character.",
                    "Functional dependency input error",
                    JOptionPane.ERROR_MESSAGE);
                return false;
            }
        }
        return true;
    }
    // R = {R1, R2, R3} input form
    else if( formtype == 2 ){

        if( data.length() > 1 ){
            JOptionPane.showMessageDialog(frame,

```

```

        "You must enter one attribute at a time.",
        "Relation Schema input error",
        JOptionPane.ERROR_MESSAGE);
    return false;
}
// production separator char '-' check
else if( data.indexOf('-') != -1 ){
    JOptionPane.showMessageDialog(frame,
        "Relation must not contain '-'",
        "Relation Schema input error",
        JOptionPane.ERROR_MESSAGE);
    return false;
}
else if( !Character.isLetterOrDigit(data.charAt(0)) ){
    JOptionPane.showMessageDialog(frame,
        "Relation must be a letter or digit",
        "Relation Schema input error",
        JOptionPane.ERROR_MESSAGE);
    return false;
}
return true;
}
// D = {D1, D2, D3}
else if( formtype == 3 ){
    // valid char check
    if( !validCharCheck(data) ){
        JOptionPane.showMessageDialog(frame,
            "Relation schema must be composed of letters and digits .",
            "Relation schema input error",
            JOptionPane.ERROR_MESSAGE);
        return false;
    }
    return true;
}
// default: no check
else{
    return true;
}
}

// *****
// valid char check of the input string
private boolean validCharCheck( String data ){
    int count = 0;
    if( formtype == 1 ){
        for( int i=0; i < data.length(); i++ ){
            // accept only one separating character
            if( data.charAt(i) == '-' ){
                if( count == 0 ){
                    count++;
                }else{
                    return false;
                }
            }
            // accept only letter or digit
            else if( !Character.isLetterOrDigit( data.charAt(i) ) ){
                return false;
            }
        }
    }
    else if( formtype == 3 ){
        for( int i=0; i < data.length(); i++ ){
            // accept only letters or digits
            if( !Character.isLetterOrDigit( data.charAt(i) ) ){
                return false;
            }
        }
    }
}

```

```

        return true;
    }

// *****
// sort a given string in ascending order
private String inOrder( String str ){
    StringBuffer buff = new StringBuffer();
    int strLen, buffLen;
    char ch;

    // get string length
    strLen = str.length();
    // for all char in the string
    for( int i=0; i < strLen; i++ ){
        // take one char
        ch = str.charAt(i);
        // get buffer length
        buffLen = buff.length();
        // find a place for the char in buffer
        int j=0;
        while( j < buffLen && buff.charAt(j) < ch){
            j++;
        }
        // add the char in right place
        buff.insert(j, ch);
    }

    // return the sorted string
    return buff.toString();
}
}

```

Figure 20: User interface for generic input form

```

/*****
File: ProgressStatus.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to the project progress status
*****/
package Project;

import java.awt.Dimension;
import java.awt.Rectangle;

import javax.swing.JFrame;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JLabel;

public class ProgressStatus{
    private static JDialog pdialog;
    private static JPanel ppanel;
    private static JLabel ptext;
    private static JProgressBar pbar;
    private static int plevel;

    public ProgressStatus( JFrame parent, String title ){
        pdialog = new JDialog( parent, title, false );

        ppanel = new JPanel();
        ppanel.setPreferredSize( new Dimension( 310, 80 ) );
        pdialog.getContentPane().add( ppanel );

        ptext = new JLabel("Starting...");
```

```

ppanel.add( ptext );
pbar = new JProgressBar();
pbar.setPreferredSize( new Dimension( 300, 20 ) );
pbar.setMinimum( 0 );
pbar.setMaximum( 20 );
pbar.setValue( 0 );
pbar.setBounds( 20, 35, 260, 20 );
ppanel.add( pbar );

pdialog.pack();
pdialog.setLocationRelativeTo(parent);
}

public static void startProgressBar( String title ){
    if( pdialog != null && !pdialog.isVisible() ){
        pdialog.setTitle( title );
        pdialog.setVisible( true );
    }
}

public static void progressBar(){
    if( pdialog != null && pdialog.isVisible() ){
        if( plevel > 19 ){
            plevel = 0;
        }else{
            plevel++;
        }

        pbar.setValue( plevel );
        Rectangle progressRect = pbar.getBounds();
        progressRect.x = 0;
        progressRect.y = 0;
        pbar.paintImmediately( progressRect );
    }
}

public static void progressText(String str){
    if( pdialog != null && pdialog.isVisible() ){
        ptext.setText( str );
        Rectangle labelRect = ptext.getBounds();
        labelRect.x = 0;
        labelRect.y = 0;
        ptext.paintImmediately( labelRect );
    }
}

public static void stopProgressBar(){
    if( pdialog != null && pdialog.isVisible() ){
        pdialog.setVisible(false);
    }
}
}

```

Figure 21: User interface for project progress status

```

/*
File: SelectProject.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to select from different type of project
***** */

package Project;

import java.awt.*;
import java.awt.event.*;

import javax.swing.JDialog;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JLabel;

// ****
// class definition
public class SelectProject extends JDialog {

    // define different type of project
    public static final int FD_CLOSURE = 111;
    public static final int A_CLOSURE = 222;
    public static final int CANONICAL_COVER = 333;
    public static final int DEPENDENCY_TEST = 444;
    public static final int BCNF_DECOMPOSITION = 555;
    public static final int TN_DECOMPOSITION = 666;
    public static final int AUTO DESIGN = 777;

    private static final String[] projects = {
        "3NF Decomposition",
        "Automated Design",
        "BCNF Decomposition",
        "Canonical Cover",
        "Closure of Attribute Sets",
        "Closure of a Set of Functional Dependencies",
        "Dependency Preservation Test",
    };

    private int projectType;

    // ****
    // constructor
    public SelectProject(Frame frame, String title) {
        super(frame, title, true);

        // project type selection combo box
        final JComboBox projectList = new JComboBox(projects);
        // label
        final JLabel label = new JLabel("Project Type:");
        // ok button
        JButton okButton = new JButton("Ok");
        // ok button event handler
        okButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                projectType = getProjectType((String) projectList.
                    getSelectedItem());
                setVisible(false);
            }
        });
        // cancel button
        JButton cancelButton = new JButton("Cancel");
        // cancel button event handler
        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

```

```

        projectType = 0;
        setVisible(false);
    });
}

// set defatult button for the window
getRootPane().setDefaultButton( cancelButton );

// arrange components
JPanel topPanel = new JPanel();
topPanel.setPreferredSize( new Dimension( 380, 50 ) );
topPanel.add(label);
topPanel.add(projectList);

// arrange components
JPanel bottomPanel = new JPanel();
bottomPanel.add(okButton);
bottomPanel.add(cancelButton);

JPanel panel = new JPanel();
panel.setPreferredSize( new Dimension( 400, 100 ) );
panel.add(topPanel, BorderLayout.CENTER);
panel.add(bottomPanel, BorderLayout.SOUTH);

//Put everything together, using the content pane's BorderLayout.
Container contentPanel = getContentPane();
contentPanel.add( panel );

pack();
}

// *****
// popup the dialog window
public int showDialog(Component comp) {

    projectType = 0;
    setLocationRelativeTo(comp);
    setVisible(true);

    return projectType;
}

// *****
// return the project type selected by the user
private int getProjectType( String projectName ){
    if( projectName.equals("") ){
        return 0;
    }
    else if( projectName.equals("3NF Decomposition") ){
        return TN_DECOMPOSITION;
    }
    else if( projectName.equals("Automated Design") ){
        return AUTO_DESIGN;
    }
    else if( projectName.equals("BCNF Decomposition") ){
        return BCNF_DECOMPOSITION;
    }
    else if( projectName.equals("Canonical Cover") ){
        return CANONICAL_COVER;
    }
    else if( projectName.equals("Closure of Attribute Sets") ){
        return A_CLOSURE;
    }
    else if( projectName.equals("Closure of a Set of Functional Dependencies") ){
        return FD_CLOSURE;
    }
    else if( projectName.equals("Dependency Preservation Test") ){
        return DEPENDENCY_TEST;
    }
}

```

```
    }
    return 0;
}
```

Figure 22: User interface to select a project type

```
*****
File: TNFDcom.java
LastUpdate: July 16, 2001
Usage: Provides an user interface to the Third Normal Form algorithm
*****  
  
package Project;  
  
import java.awt.*;
import java.awt.event.*;  
  
import java.util.TreeSet;
import java.util.Iterator;  
  
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;  
  
import Project.Algorithms.TNF;
import Project.BasicProject;  
  
*****  
// class definition
public class TNFDcom extends BasicProject implements ActionListener{  
  
    // input space
    private TreeSet f, r;  
  
    // output space
    private TreeSet fp;  
  
    // input form variable
    private InputForm iff, ifr;  
  
    // algorithm
    private TNF tnf;  
  
    // constructor, initialization
    public TNFDcom(JFrame mainWindow,
                   JMenuBar mainMenuBar, JTextArea mainWindowOutput ){
        super(mainWindow, mainMenuBar, mainWindowOutput );  
  
        // menu to the menu bar
        JMenu menu = new JMenu("3NF Decomposition");
        menuBar.add(menu);  
  
        // add "Input: R - Relation Schema" menu item
        JMenuItem menuItem = new JMenuItem("Input: R - Relation Schema");
        menuItem.addActionListener(this);
        menu.add(menuItem);  
  
        // add "Input: F - Set of Functional Dependencies" menu item
        menuItem = new
                    JMenuItem("Input: F - Set of Functional Dependencies");
```

```

menuItem.addActionListener(this);
menu.add(menuItem);

menu.addSeparator();

// add "Compile" menu item
menuItem = new JMenuItem("Compile");
menuItem.addActionListener(this);
menuItem.setEnabled(false);
menu.add(menuItem);

// initialize Relation Schema input windows
ifr = new InputForm( window, "Input: R - Relation Schema", 3);

// initialize Functional Dependency input windows
iff = new InputForm( window,
                     "Input: F - Set of Functional Dependencies", 1);

// initialize input space
f = new TreeSet();
r = new TreeSet();

// initialize algorithm
tnf = new TNF();

// update output window
output();
}

// *****
// menu event action handler
public void actionPerformed(ActionEvent e) {
    // get event source menu item
    JMenuItem source = (JMenuItem)(e.getSource());

    // get event source menu item text
    String action = source.getText();

    // perform action based on the selected menu option
    if( action.equals("Input: R - Relation Schema") ){
        // relation schema input window
        rInputWindow();
    }
    else if(
        action.equals("Input: F - Set of Functional Dependencies") ){
        // functional dependency input window
        fInputWindow();
    }
    else if( action.equals("Compile") ){
        fp = tnf.compile( r, f );
        // compile not needed
        recompile = false;
        // not saved
        saveProject = true;
        // update output window
        output();
    }
}

// *****
// relation schema input form window
private void rInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = ifr.showDialog( window, r );

    // check if input has changed
    if( changed ){

}

```

```

        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

}

// *****
// functional dependency input form window
private void fInputWindow(){
    // variable to keep track whether the input
    // has changed since last compile
    boolean changed;

    // take input from user
    changed = iff.showDialog( window, f );

    // check if input has changed
    if( changed ){
        // recompile needed
        recompile = true;
        // not saved
        saveProject = true;
        // update output window
        output();
    }

    // enable/disable compile menu option
    if( f.isEmpty() ){
        menuBar.getMenu(1).getItem(3).setEnabled(false);
    }else{
        menuBar.getMenu(1).getItem(3).setEnabled(true);
    }
}

// *****
// print to application output window
private void output(){
    StringBuffer sb = new StringBuffer();

    // print project title
    sb.append("[Project]\n");
    sb.append("3NF Decomposition\n\n");

    // print input R
    if( !r.isEmpty() ){
        sb.append("[Input: R - Relation Schema]\n");
        sb.append( TreeSetToString( r ) + "\n" );
    }

    // print input F
    if( !f.isEmpty() ){
        sb.append("[Input: F - Set of Functional Dependencies]\n");
        sb.append( TreeSetToString( f ) + "\n" );
    }

    // print output
    if( recompile == false ){
        sb.append("[Output: 3NF Decomposition of R]\n");
        sb.append( TreeSetToString( fp ) + "\n" );
    }

    // print ot application output window
    output.setText( sb.toString() );
}
}

```

Figure 23: User interface for 3NF decomposition algorithm

```

*****  

File: MainWindow.java  

LastUpdate: November 21, 2001  

Usage: Application window for various database design algorithm demonstration  

*****  

import java.awt.*;  

import java.awt.event.*;  

import javax.swing.JMenu;  

import javax.swing.JMenuItem;  

import javax.swing.JMenuBar;  

import javax.swing.KeyStroke;  

import javax.swing.JTextArea;  

import javax.swing.JScrollPane;  

import javax.swing.JFrame;  

import java.awt.GraphicsConfiguration;  

import javax.swing.UIManager;  

import Project.SelectProject;  

import Project.ProgressStatus;  

import Project.BasicProject;  

import Project.FDClosure;  

import Project.AClosure;  

import Project.DTest;  

import Project.CCover;  

import Project.BCNFDcom;  

import Project.TNFDcom;  

import Project.AutoDesign;  

// class definition *****  

public class MainWindow extends JFrame implements ActionListener {  

    // application output window  

    JTextArea output;  

    // scroll bar of the output window  

    JScrollPane scrollPane;  

    // application menu bar  

    JMenuBar menuBar;  

    // menu of the menu bar  

    JMenu menu;  

    // menu item of the menu  

    JMenuItem menuItem;  

    // project type selection window  

    SelectProject sp;  

    // current project reference  

    Object project;  

    // current project type  

    int projectType;  

    // constructor *****  

    public MainWindow() {  

        // application window event handler  

        addWindowListener(new WindowAdapter() {  

            public void windowClosing(WindowEvent e) {  

                windowExit();  

            }
        });
  

        // add components to the window  

        Container contentPane = getContentPane();  

        output = new JTextArea(5, 30);  

        output.setEditable(false);  

        scrollPane = new JScrollPane(output);  

        contentPane.add(scrollPane, BorderLayout.CENTER);  

        // create the menu bar  

        menuBar = new JMenuBar();

```

```

        setJMenuBar(menuBar);

        // add "File" menu
        menu = new JMenu("File");
        menu.setMnemonic(KeyEvent.VK_F);
        menuBar.add(menu);

        // add "New Project" menu item in "File"
        menuItem = new JMenuItem("New Project", KeyEvent.VK_N);
        menuItem.setAccelerator(KeyboardStroke.getKeyStroke(KeyEvent.VK_N,
            ActionEvent.ALT_MASK));
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // add "Save Project" menu item in "File"
        menuItem = new JMenuItem("Save Project", KeyEvent.VK_S);
        menuItem.setAccelerator(KeyboardStroke.getKeyStroke(KeyEvent.VK_S,
            ActionEvent.ALT_MASK));
        menuItem.setEnabled(false);
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // add "Close Project" menu item in "File"
        menuItem = new JMenuItem("Close Project", KeyEvent.VK_C);
        menuItem.setAccelerator(KeyboardStroke.getKeyStroke(KeyEvent.VK_C,
            ActionEvent.ALT_MASK));
        menuItem.setEnabled(false);
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // add menu item separator
        menu.addSeparator();

        // add "Exit" menu item in "File"
        menuItem = new JMenuItem("Exit", KeyEvent.VK_X);
        menuItem.setAccelerator(KeyboardStroke.getKeyStroke(KeyEvent.VK_X,
            ActionEvent.ALT_MASK));
        menuItem.addActionListener(this);
        menu.add(menuItem);

        // instantiate new project selection window
        sp = new SelectProject( this, "New Project" );
    }

    // menu event handler *****
    public void actionPerformed(ActionEvent e) {
        // get the event source and text
        JMenuItem source = (JMenuItem)e.getSource();
        String action = source.getText();

        // perform action based on the event
        if( action.equals("New Project") ){
            newProject();
        }
        else if( action.equals("Save Project") ){
            saveProject();
        }
        else if( action.equals("Close Project") ){
            closeProject();
        }
        else if( action.equals("Exit") ){
            windowExit();
        }
    }

    // open new project *****
    private void newProject(){
        // bring up the project selection dialog

```

```

        int newProjectType = sp.showDialog( this );

        // if user have selected valid project type, open new project
        if( newProjectType != 0 ){

            // close opened project
            if( project != null ){
                closeProject();
            }

            // create new project
            switch( newProjectType ){
                case SelectProject.FD_CLOSURE:
                    project = new FDClosure(this, menuBar, output);
                    projectType = SelectProject.FD_CLOSURE;
                    break;

                case SelectProject.A_CLOSURE:
                    project = new AClosure(this, menuBar, output);
                    projectType = SelectProject.A_CLOSURE;
                    break;

                case SelectProject.CANONICAL_COVER:
                    project = new CCover(this, menuBar, output);
                    projectType = SelectProject.CANONICAL_COVER;
                    break;

                case SelectProject.DEPENDENCY_TEST:
                    project = new DTest(this, menuBar, output);
                    projectType = SelectProject.DEPENDENCY_TEST;
                    break;

                case SelectProject.BCNF_DECOMPOSITION:
                    project = new BCNFDecom(this, menuBar, output);
                    projectType = SelectProject.BCNF_DECOMPOSITION;
                    break;

                case SelectProject.TN_DECOMPOSITION:
                    project = new TNFDcom(this, menuBar, output);
                    projectType = SelectProject.TN_DECOMPOSITION;
                    break;

                case SelectProject.AUTO DESIGN:
                    project = new AutoDesign(this, menuBar, output);
                    projectType = SelectProject.AUTO DESIGN;
                    break;
            }

            // set menu options
            if( project != null ){
                // set "Save Project" menu option visible
                menuBar.getMenu(0).getItem(1).setEnabled(true);
                // set "Close Project" menu option visible
                menuBar.getMenu(0).getItem(2).setEnabled(true);
                // update view
                setVisible( true );
            }
        }

        // save current project *****
    private void saveProject(){
        switch( projectType ){
            case SelectProject.FD_CLOSURE:
                ((FDClosure) project).save();
                break;

            case SelectProject.A_CLOSURE:
                ((AClosure) project).save();
                break;
        }
    }
}

```

```

        case SelectProject.CANONICAL_COVER:
            ((CCover) project).save();
            break;

        case SelectProject.DEPENDENCY_TEST:
            ((DTest) project).save();
            break;

        case SelectProject.BCNF_DECOMPOSITION:
            ((BCNFDcom) project).save();
            break;

        case SelectProject.TN_DECOMPOSITION:
            ((TNFDcom) project).save();
            break;

        case SelectProject.AUTO DESIGN:
            ((AutoDesign) project).save();
            break;
    }
}

// close current project *****
private void closeProject(){
    switch( projectType ){
        case SelectProject.FD_CLOSURE:
            ((FDclosure) project).close();
            break;

        case SelectProject.A_CLOSURE:
            ((AClosure) project).close();
            break;

        case SelectProject.CANONICAL_COVER:
            ((CCover) project).close();
            break;

        case SelectProject.DEPENDENCY_TEST:
            ((DTest) project).close();
            break;

        case SelectProject.BCNF_DECOMPOSITION:
            ((BCNFDcom) project).close();
            break;

        case SelectProject.TN_DECOMPOSITION:
            ((TNFDcom) project).close();
            break;

        case SelectProject.AUTO DESIGN:
            ((AutoDesign) project).close();
            break;
    }

    // set "Save Project" menu option
    menuBar.getMenu(0).getItem(1).setEnabled(false);
    // set "Close Project" menu option
    menuBar.getMenu(0).getItem(2).setEnabled(false);
    // update view
    repaint();

    // update variables
    project = null;
    projectType = 0;
}

// application entry point *****
public static void main(String[] args) {

```

```
// set system default window appearance
try {
    UIManager.setLookAndFeel(
        UIManager.getSystemLookAndFeelClassName());
} catch (Exception e) { }

// create application window
MainWindow main_window = new MainWindow();
// set title
main_window.setTitle("Automated Database Design");
// set application window size to full screen
main_window.setBounds( GraphicsEnvironment.
    getLocalGraphicsEnvironment().getDefaultScreenDevice().
    getDefaultConfiguration().getBounds() );
// update view
main_window.setVisible(true);
}

// application window exit event handler *****
private void windowExit(){

    // if there is any open project
    if( project != null ){
        // close project
        closeProject();
    }

    // exit system
    System.exit(0);
}
```

Figure 24: Application window for all the algorithms

The screenshot shows a Windows-style application window titled "Instructor". It contains four text input fields: "Id" (empty), "Name" (Nafur Rahman), "Email" (nafur@yahoo.com), and "Password" (XXXXXXXX). Below the fields is a status bar with the text "Record: 1 < < | 2 > >> \* of 3".

Figure 25: SIS Instructor form

The screenshot shows a Windows-style application window titled "Course". It contains four text input fields: "Id" (empty), "Title" (Advanced Programming in Visual Basic), "Prefix" (CIS), and "Number" (33262). Below the fields is a status bar with the text "Record: 1 < < | 1 > >> \* of 5".

Figure 26: SIS Course form

The screenshot shows a Windows-style application window titled "Student". It contains four text input fields: "Id" (1226), "Name" (Peter D Pesa), "Email" (peter@pesa.com), and "Password" (XXXXXXXXXX). To the right of the "Id" field is a dropdown menu labeled "Start Semester" with the value "Fall 1887". Below the fields is a status bar with the text "Record: 1 < < | 2 > >> \* of 24".

Figure 27: SIS Student form

**Offered Courses**

Id	▼		
Semester	Fall 2001		
Course Id	Advanced Programming in Visual Basic		
Instructor Id	Nafiu Rahman		
Section	1	Credit Hour	3

Record: **1** of 5

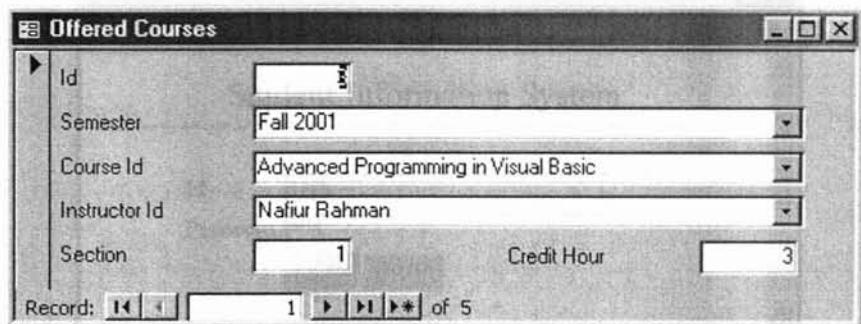


Figure 28: SIS Offer form

**Course Enrollment**

Semester:	Fall 2001	
Course:	Advanced Programming in Visual Basic	
Student Id	Grade	Pass
Robert A Bellard	X	<input type="checkbox"/>
Tameka L Clayton	X	<input type="checkbox"/>
Kienia Markia Green	X	<input type="checkbox"/>
Romell Harper	X	<input type="checkbox"/>
Tania R Miller	X	<input type="checkbox"/>
Peter D Pesa	X	<input type="checkbox"/>
Rubayet Salam	X	<input type="checkbox"/>
Marja M Smith	X	<input type="checkbox"/>
Donald Whaley	X	<input type="checkbox"/>
*		

Record: **1** of 9

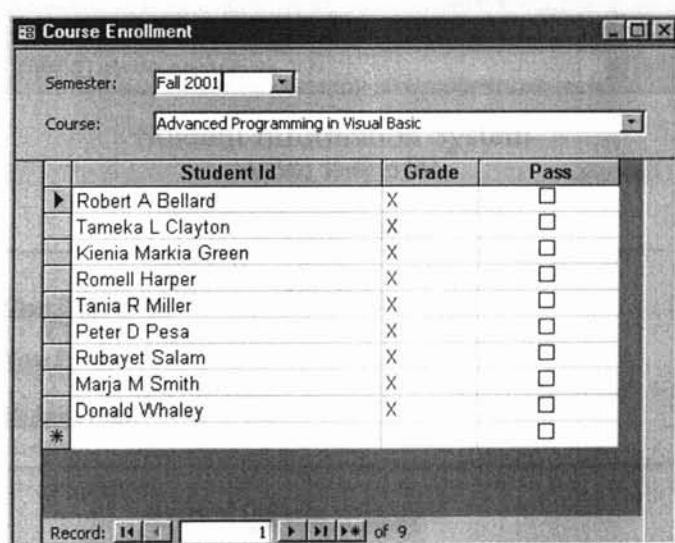


Figure 29: SIS Enrollment form

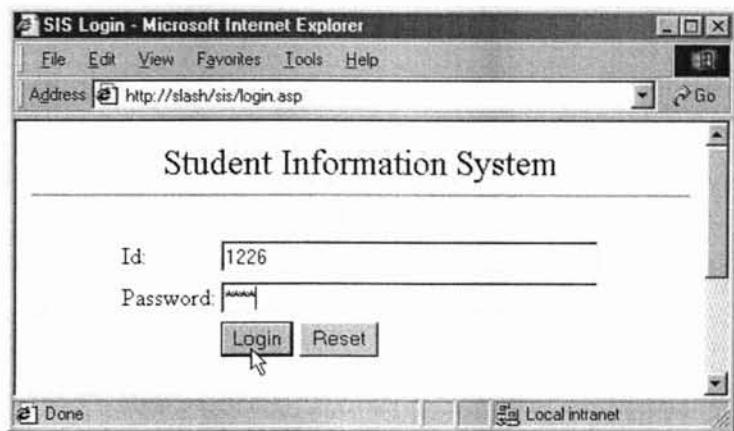


Figure 30: SIS Login screen

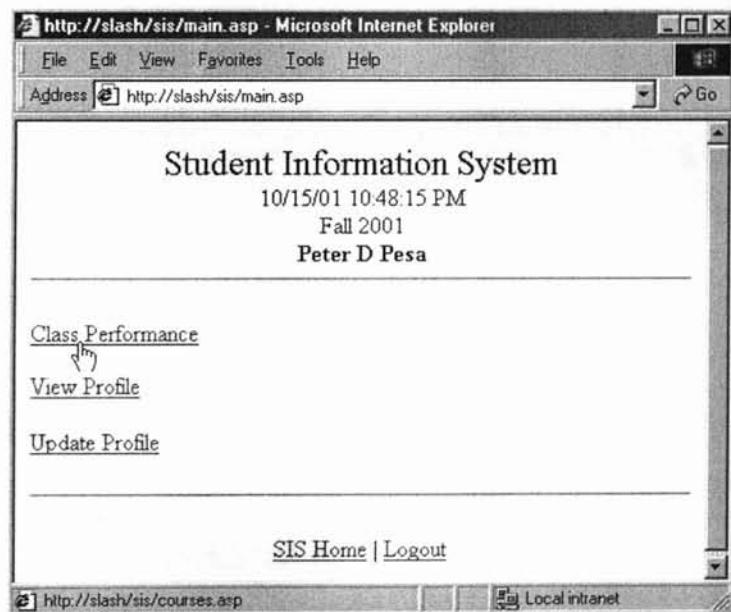


Figure 31: SIS Student home screen

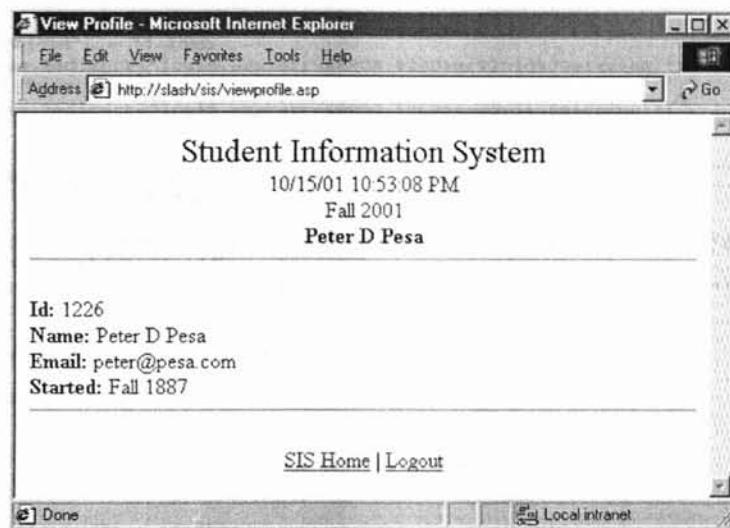


Figure 32: SIS View student profile screen

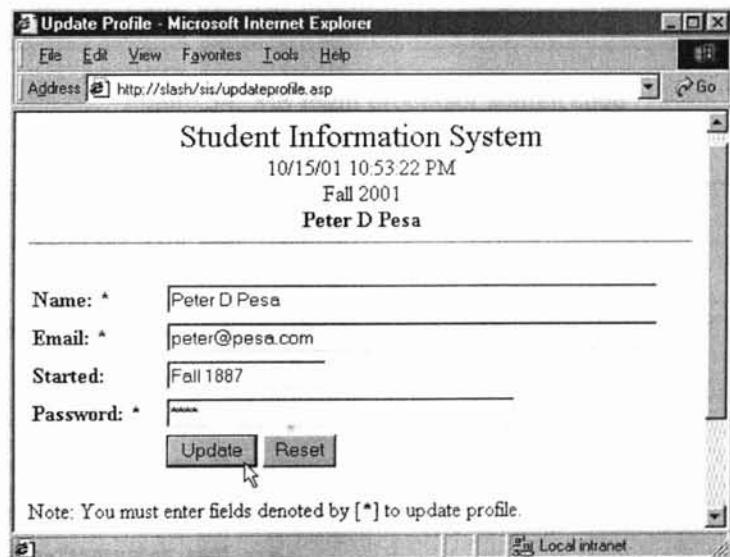


Figure 33: SIS Update student profile screen

```

<%@ Language=VBScript %>

<%
If Request.Form("id") = "" Then
    Response.Redirect("login.asp?err=ERROR:%20User%20id%20missing!")
Elseif Request.Form("pass") = "" Then
    Response.Redirect("login.asp?err=ERROR:%20Password%20missing!")
Else

    id = Request.form("id")
    pass = Request.form("pass")

    strProvider = "DSN=" & Application("dsn") & ";UID=" & Application("dbuser") &
    ";PWD=" & Application("dbpass") & ";"
    strQuery = "SELECT Name, Locked FROM Student WHERE (Id ='" & id & "') AND
    (Password=''" & pass & "');"
    Set rst = Server.CreateObject("ADODB.Recordset")
    rst.Open strQuery, strProvider

    ON ERROR RESUME NEXT

    If rst.EOF Then
        Response.Redirect("login.asp?err=ERROR:%20Login%20failed!")
    End If

    If rst("Locked") = -1 Then
        rst.Close
        Response.Redirect("login.asp?err=ERROR:%20Your%20account%20is%20locked!")
    End If

    Session("id") = id
    Session("name") = rst("Name")
    Session("db") = strProvider
    Response.Redirect("main.asp")

End If

%>

```

Figure 34: SIS login processor source code

```

<%@ Language=VBScript %>

<!--#include file="checklogin.asp" -->

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 4.0">
</HEAD>
<BODY>

<!--#include file="titlebar.asp" -->

<P><A HREF="courses.asp">Class Performance</A></P>
<P><a href="viewprofile.asp">View Profile</a></P>
<P><a href="updateprofile.asp">Update Profile</a></P>

<!--#include file="statusbar.asp" -->

</BODY>
</HTML>

```

Figure 35: SIS home page source code

```
<%
If Session("id") = "" Then
    Response.Redirect("login.asp")
End If
%>
```

Figure 36: SIS check login source code

```
<p align="center">
<big><big>Student Information System</big></big><br>
<%= Now() %><br>
<%= Application("semester") %><br>
<b><%= Session("name") %></b>
<hr>
</p>
```

Figure 37: SIS title bar source code

```
<hr>
<p align="center">
<a href="main.asp">SIS Home</a> | <a href="logout.asp">Logout</a>
</p>
```

Figure 38: SIS status bar source code

```
<%
Set rst = Server.CreateObject("ADODB.Recordset")
%>
```

Figure 39: SIS open database connection source code

```
<%
rst.Close
Set rst = Nothing
%>
```

Figure 40: SIS close database connection source code

```
<!--#include file="checklogin.asp" -->
<!--#include file="rstopen.asp" -->

<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>View Profile</title>
</head>

<body>

<!--#include file="titlebar.asp" -->

<%
strQuery = "SELECT * FROM Student WHERE Id=' " & Session("id") & "' ;
rst.Open strQuery, Session("db")
%>

<% If Not rst.EOF Then %>
```

```
<b>Id:</b> <%= Session("id") %><br>
<b>Name:</b> <%= Session("name") %><br>
<b>Email:</b> <%= rst("Email") %><br>
<b>Started:</b> <%= rst("Start Semester") %><br>
<% End If %>

<!--#include file="statusbar.asp" -->

</body>

</html>

<!--#include file="rstclose.asp" -->
```

Figure 41: SIS view profile source code

```
<Script Language="vbscript" Runat="server">

Sub Application_OnStart()
    Application("dsn") = "sis"
    Application("dbuser") = "students"
    Application("dbpass") = "students"
    Application("semester") = "Fall 2001"
    Application("uploadaddir") = "D:\Inetpub\SIS\upload\
End Sub

Sub Session_OnStart()
    Response.Redirect("login.asp")
End Sub

Sub Session_OnEnd()
End Sub

Sub Application_OnEnd()
End Sub

</script>
```

Figure 42: SIS global.asa file

2

VITA

Nafiur Rahman Khandaker Mohammad

Candidate for the Degree of

Master of Science

Thesis: A SEMI-AUTOMATED PROCEDURE FOR SMALL RELATIONAL  
DATABASE LOGICAL DESIGN

Major Field: Computer Science

Biographical:

Personal Data: Born in Bogra, Bangladesh on December 29, 1973, the only son of Nadira Khandaker and Idris Ali Khandaker.

Education: Graduated from Computer Science Department, North South University, Dhaka, Bangladesh in August, 1996; received Bachelor of Science degree in Computer Science. Completed the requirements of the Master of Science degree at Oklahoma State University in December 2001.

Professional Experience: Jan 1997–Dec 1999: Programmer  
Hongkong & Shanghai Banking Corporation, Dhaka, Bangladesh  
Jan 2000–Aug 2001: Research Assistant (Web Application Developer)  
Department of Political Science, Oklahoma State University  
Aug 2001–Present: Instructor  
Computer Science Department, Wiley College, Marshall, Texas