

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, Mi 48106-1346 USA
800-521-0600**

UMI[®]

NOTE TO USERS

Page(s) not included in the original manuscript are unavailable from the author or university. The manuscript was microfilmed as received.

46

This reproduction is the best copy available.

UMI[®]

ENHANCED WEB BASED CINDI SYSTEM

YUHUI WANG

A MAJOR REPORT

IN

DEPARTMENT OF COMPUTER SCIENCE

**PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA**

AUGUST 2002

© YUHUI WANG, 2002



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file / Votre référence

Our file / Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-72947-8

Canada

Abstract

Enhanced Web Based CINDI System

Yuhui Wang

The web-based CINDI (Concordia INdexing and DIsccovery) System attempts to solve the common problem of the current search engines, such as lack of a standard indexing and an information query interface. It proposes Semantic Header as the standard index scheme, and stores the index entry into MySQL database management system. By invoking a set of graphic user interface, the CINDI system allows the resource contributor to catalog his own resource and enables the user to search for hypermedia documents based on the title, author(s), keyword(s) and subject search criteria.

This report describes the redesign and implementation of the Semantic Header database, and the design and implementation of the graphic user interface for the resource registration subsystem and resource search subsystem. The user-friendly interface has been implemented with PHP script language on the Linux platform. The three-tier client/server architecture is used in the web-based CINDI system, with Apache web server exchanging information between the web Browser (the client) and the backend MySQL database.

Acknowledgements

I express deep gratitude to my supervisor, Professor Bipin C. DESAI, for his encouragement and invaluable discussion. Like many others, I consider myself blessed to be under his supervision. He was always gracious in giving me the time even amidst tight schedule.

I am thankful to Prof. Dhrubajyoti Goswami for his guidance and help.

Furthermore, I would also like to thank Zhan Zhang, one member of the CINDI project, help me integrate ASHG with the whole system.

Last but not the least, I express deep veneration for my dearest parents for teaching me the importance of good education, from which everything else springs.

Contents

1 Introduction	1
1.1 Existing Problem of Search Engine	1
1.2 CINDI System.....	3
1.3 Organization of the Report.....	6
2 Architecture of the CINDI System	7
2.1 Client/Server Architecture	7
2.1.1 Two Tier Client/Server Architecture	7
2.1.2 Three Tier Client/Server Architecture	8
2.1.3 CINDI System Architecture	10
2.2 MYSQL Database	12
2.2.1 Introduction	12
2.2.2 MySQL Features.....	13
2.2.3 InnoDB Tables.....	15
2.3 Sub-System Design	16
3 Semantic Header Database System	19
3.1 Semantic Header Structure.....	19
3.2 Entity-Relationship Data Model.....	21
3.3 Database Table Description	23
3.3.1 Resource Table	23
3.3.2 Subject Table.....	25
3.3.3 Resource_subject Table.....	25
3.3.4 Author Table.....	26
3.3.5 Resource_author Table.....	27
3.3.6 Users Table.....	27
3.3.7 Annotation Table.....	28
3.3.8 Contributor Table	28
3.3.9 Language Table	29
3.3.10 Coverage Table.....	29
3.3.11 System_req Table	30
3.3.12 Identifier Table	31
3.3.13 Classification Table.....	31
3.4 Primary and Foreign Keys	32
3.5 Intermediate Tables of ASHG Subsystem	33
3.6 Tuning Performance.....	37
3.6.1 Using Indexes	37
3.6.2 Streamline SQL Statement	38
3.6.3 Using Persistent Connection.....	39
3.6.4 Minimize Transaction.....	39
4 Security Control Sub-system.....	43
4.1 Apache SSL Server	43

4.2	User/Contributor Registration.....	44
4.3	Login	46
4.4	Session Handling.....	48
5	Resource Registration Sub-system	50
5.1	Manual Resource Registration	51
5.2	Automatic Resource Registration.....	53
5.3	Author Registration.....	60
6	Search and Annotation Sub-systems	62
6.1	Search Query Structure	63
6.2	Simple Search.....	64
6.3	Intermediate Search.....	65
6.4	Advanced Search.....	67
6.5	Annotation Sub-system	70
6.6	Error Handling for Search Sub-system	71
7	Conclusion and Future Work	73
7.1	Conclusion	73
7.2	Contribution of this Report	74
7.3	Future Work	75
	References	78

List of Figures

Figure 2.1.1: Two tier client/server architecture design.....	7
Figure 2.1.2: Three tier distributed client/server architecture design	9
Figure 2.1.3: Three tier CINDI system architecture.....	11
Figure 3.2: ER diagram of the Semantic Header Database.....	22
Figure 4.2.1: User registration interface.....	45
Figure 4.2.2: Error handling of the user registration.....	46
Figure 4.3: User login interface.....	47
Figure 4.4: Warning page for illegal user	49
Figure 5: Resource registration sub-system interface	50
Figure 5.1: Manual resource registration interface (Part 1)	52
Figure 5.1: Manual resource registration interface (Part 2)	53
Figure 5.2.1: Automatic Resource Registration Interface (Part 1).....	55
Figure 5.2.1: Automatic Resource Registration Interface (Part 2).....	56
Figure 5.2.2: Automatic subject registration Interface.....	59
Figure 5.2.3: Automatic subject registration Interface.....	60
Figure 5.3: Resource author registration interface	61
Figure 6: Search sub-system interface	62
Figure 6.2: Simple resource search interface	65
Figure 6.3: Intermediate resource search interface	67
Figure 6.4.1: Advanced resource search interface	69
Figure 6.4.2: Resource search results.....	70
Figure 6.5: Annotation subsystem interface.....	71
Figure 6.6.1: No search criteria error	72
Figure 6.6.2: Query Boolean condition error	72

List of Tables

Table 1: Sample Search Statistics for searching on Bipin (AND) Desai	2
Table 2: Comparison: Reading 2,000,000 rows by index	14
Table 3: Comparison: Inserting (350,768) rows	14
Table 4: Primary key and foreign key for each table	33

Chapter 1

Introduction

1.1 Existing Problem of Search Engine

In recent years, an increasing number of people are publishing their articles, reports and other information resources on the Internet using the World Wide Web, which has permitted Internet to become a popular repository of information. There needs an efficient way to support local as well as the remote search and retrieval of information stored on the web site. Virtual library (or digital library) is built to collect texts, books, journals, newspapers, national directories of various types, sound, images, scientific data etc. in electronic form. By invoking the web, users can access this distributed information system from anywhere in the world. In practice, the digital library consists of several components: a method of collecting information from various format files, creating catalog and indices about these resources, a database for storing these information and a method to search and manipulate the database effectively, a graphical interface to interact with users.

In order to make effective use of the wealth of information on the Internet, many search engines such as AltaVista, Yahoo, Google, HotBot and Lycos, have been developed and some of them have become very popular. Google catalogs the collected information by its own indexing system, which generates index manually or by robots. In

spite of the fact that the same cataloging system is used, the same item may be differently catalogued or classified in two different libraries [1]. Since many of these systems attempt to match the specified search terms (or keywords) without regard for the context in which the words appear in the target information resource, it can cause miss-hits and missed items. For example, a test was conducted in February 2001 on the existing search systems by using keywords **Bipin (AND) Desai**. There were totally 329 URL (Universal Resource Locator) containing test terms on the WWW at that time. The test results (see Table 1) show that none of these systems is always successful in retrieving the documents sought [2]. The terminology used in Table 1 is as follows:

- Number of Hits: The number of document found containing Bipin (AND) Desai.
- Number of Duplicates: The number of times the same document was in the search result, i.e. the same document being retrieved from more than one site.
- Number of Miss-hits: The number of irrelevant documents found.
- Number of Items missed: The number of documents not found even though they existed on the WWW.

Search Sytem	Number of Hits	Number of Duplicates	Number of Miss-hits	Number of Items missed
AltaVista	99	24	67	230
Google	155	10	403	174
HotBot	62	21	121	267
Lycos	239	37	711	90

Table 1: Sample Search Statistics for searching on Bipin (AND) Desai

In summary, we list several drawbacks of the existing search systems:

- Lack of standard: the same item may be differently catalogued or classified by two different search engines.

- Resources are usually referenced by words or phrases instead of by context such as title, author, subject etc. The majority of searches begin with a title, name of one of the authors (70%), subject and sub-subject (50%) [1].
- Many systems do not provide abstract or annotation about a document or file. Usually the user has to read the actual resource to decide whether it meets the needs.
- Indexing systems are commonly used to find relevant information, but most of them increase the traffic on the network, and do not coordinate and share gathered information among each other.

1.2 CINDI System

The above problems could be avoided by starting with a standard index structure and building a bibliographic system using standardized control definitions. Since the purpose of index and bibliography is to allow easy access to the original materials, it has to be accurate, easy to use (usage via title, author, subject etc.), properly classified, up-to-date and complete for its area of coverage. In the Internet distributed system, a semi-automatic mechanism can be used to scan the published works and assign each work to appropriate subjects, then the providers of the resources will validate or update the result. Whereas an alternate scheme let the provider prepare and enter the bibliographic information about each resource using the standardized index scheme.

The objective of CINDI (Concordia INdexing and DIsccovery) System, which is proposed by Desai et al [3], is to build a system that enables any resource contributor to catalog his own resource and any user to subsequently search for them using a typical

search item such as Author, Title, Subject, etc. The system will offer a bibliographic database that provides information about documents available on the Internet. We can divide the CINDI system into four main sub-systems:

- A distributed and replicated database system to store the simple index structure that describes each information resource.
- A registering system is required to facilitate the provider (author/creator) of a resource to register the bibliographic information about the resource.
- A search system allows users to enter a query based on multiple fields.
- An annotation system to collect the readers' comments on the resource.

Web-based CINDI system provides a set of graphic user interfaces: a user can register/update an index entry, make annotations to any index entry, and execute query for information discovery using a Web Browser.

A standardized index scheme, i.e. Semantic Header [4], is designed to ensure homogeneity of the syntax and semantics of such an index. It is the heart of the system that records the characteristics of the source data and provides the succinct information about the source data. The intent of the semantic header is to include those items that are most often used in the search of the information resource, for instance, the title, author, subject and sub-subject field. The abstract field of the semantic header allows users to make better decisions regarding the relevance of the source resource. Since the provider of the resource is responsible for preparing the index information, such index entry has high reliability. These index entries are stored in a distributed database system called the Semantic Header Distributed Database System (SHDDB). The distribution is based on subject areas, and the database can be horizontally partitioned. The replicated nature of

the database ensures distribution of load and continued access to the bibliography when one or more sites are temporarily nonfunctional.

Once the user has entered a search request, the server side process will communicate with the nearest SHDDB catalogue to determine the appropriate site of the SHDDB database. Then, the server process connects with this database and retrieves one or more semantic headers. Finally, the result of the query will be collected and sent to the user, he/she may select one of the entries from the search results to view the complete meta-data (i.e. semantic header). The system will increase the meta-data access counter by one. The user can also read the pertinent source resource by using a web browser. One of the major future tasks is to design and implement a distributed database. However, in this report, we just built a centralized database.

The index registering sub-system interface allows the provider to enter the information with the help of the knowledge-based expert system. After the information is correctly entered, the author can decide to register the semantic header entry to the SHDDB database. When the header information is accepted by the database, the author/creator will be notified. The proposed system also provides a semi-automatic mechanism, Automatic Semantic Header Generator (ASHG), a program package to generate a draft version of the semantic header. ASHG subsystem can extract the semantic header information (such as title, author, subject, keywords, abstract, annotation) from many different file formats like HTML, TEXT, LATEX and RTF.

1.3 Organization of the Report

This report describes the architecture of the CINDI system, the redesign and implementation of the database system and the interface between the CINDI database system and the Web Browser.

Chapter 2 presents the details of the three-tier client/server architecture of the CINDI system in the first section. MySQL database is covered in section 2.2, and InnoDB table is introduced to support the foreign key constraint and handle the transaction. Section 2.3 gives a brief overview of each subsystem and their relationship.

The details of the design and implementation of the Semantic Header database is described in Chapter 3. Section 3.1 shows the structure of the Semantic Header. Each table and its corresponding primary/foreign key are discussed in section 3.3 and section 3.4. Section 3.5 describes the intermediate tables that are used to integrate with the Automatic Semantic Header Generator subsystem.

The following chapters describe the graphical user interface of the web application. Chapter 4 shows the security control sub-system such as login and session handling. Manual and automatic resource registration are presented in Chapter 5. Chapter 6 discusses the search sub-system and the annotation sub-system. Finally in Chapter 7, the conclusion is drawn and suggestions are provided for future work.

Chapter 2

Architecture of the CINDI System

2.1 Client/Server Architecture

2.1.1 Two Tier Client/Server Architecture

Two tier client/server architecture consists of three components distributed in two layers: client (requester of services) and server (provider of services). It was developed in the 1980s from the file sharing architecture design. The three components are:

- User System Interface (such as session, text input, dialog, and display management services)
- Processing Management (such as process development, process enactment, process monitoring, and process resource services)
- Database Management (such as data and file services)

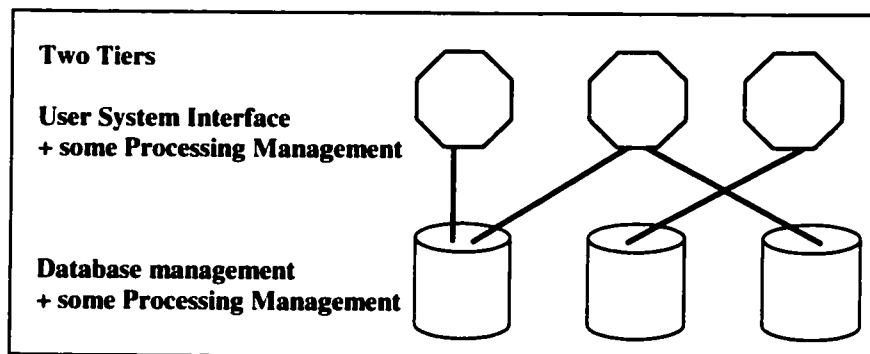


Figure 2.1.1: Two tier client/server architecture design

Figure 2.1.1 depicts the two-tier client/server architecture, which allocates the user system interface exclusively to the client, places database management on the server and splits the processing management between client and server, creating two layers.

In general, the user system interface client invokes services from the database management server. In many two-tier designs, most of the application portion of processing is in the client environment. The database management server usually provides the portion of the processing related to accessing data (often implemented in store procedures). Clients commonly communicate with the server through SQL statements or a call-level interface.

The two-tier client/server architecture is a good solution for distributed computing when work groups are defined as a dozen to 100 people interacting on a LAN simultaneously. But when the number of users exceeds 100, performance begins to deteriorate. The reason is that the server maintains a connection via "keep-alive" messages with each client, even when no work is being done. A second limitation of the two-tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications.

2.1.2 Three Tier Client/Server Architecture

The three tier architecture (also referred to as the multi-tier architecture) emerged in 1990s to overcome the limitations of the two tier architecture. In the three-tier architecture, a middle tier was added between the user system interface client

environment and the database management server environment. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two-tier architecture) by providing functions such as queuing, application execution, and database staging.

A three tier distributed client/server architecture (as shown in Figure 2.1.2) includes a user system interface at the top tier in which user services reside. The third tier provides database management functionality and is dedicated to data and file services that can be optimized without using any database management system languages. The data management component ensures that the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication.

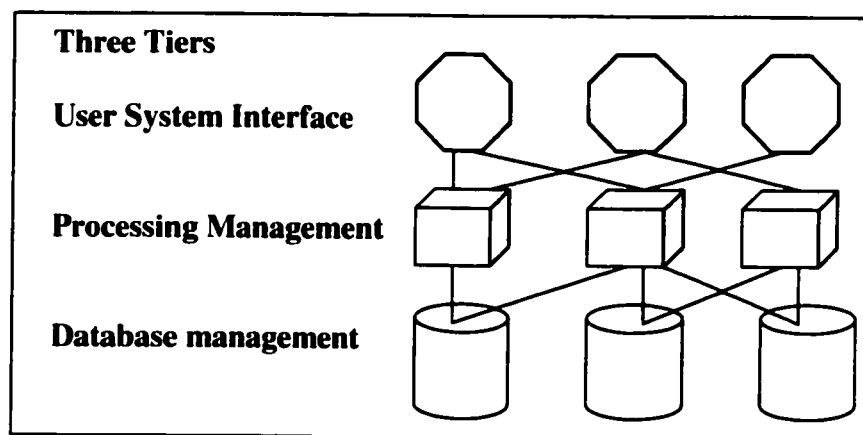


Figure 2.1.2: Three tier distributed client/server architecture design

The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) that are shared by multiple applications. There are a variety of ways of implementing this middle tier, such

as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client.

The three tier architecture is used when an effective distributed client/server design is needed that provides increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications [7].

Sometimes, the middle tier is divided in two or more unit with different functions, in these cases the architecture is often referred as multi layer. For example, some Internet applications typically have light clients written in HTML and application servers written in C++ or Java, the gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This layer receives requests from the Internet clients and generates html using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic.

2.1.3 CINDI System Architecture

The web-based CINDI system is a three-tier Internet application (as shown in Figure 2.1.3), which supports the multi-user, multi-threaded environment. Clients access the

web site by specifying the web address in a web browser and the web browser communicates with the web server through the HTTP protocol. The Apache web server runs PHP scripts and returns a dynamic web page to the web browser based on the user requirements. If the request needs data from the database, the web based middleware will be employed to interact with the backend MySQL database management system. PHP scripts play the same roles as any other CGI programs, such as collecting form data, generating dynamic page content, maintaining the session, or sending/receiving cookie. One significant feature of PHP is its support for a wide range of databases such as MySQL, Oracle, dBase, PostgreSQL, Sybase, IBM DB2, Informix etc [8], which make writing a database-enabled web page easy.

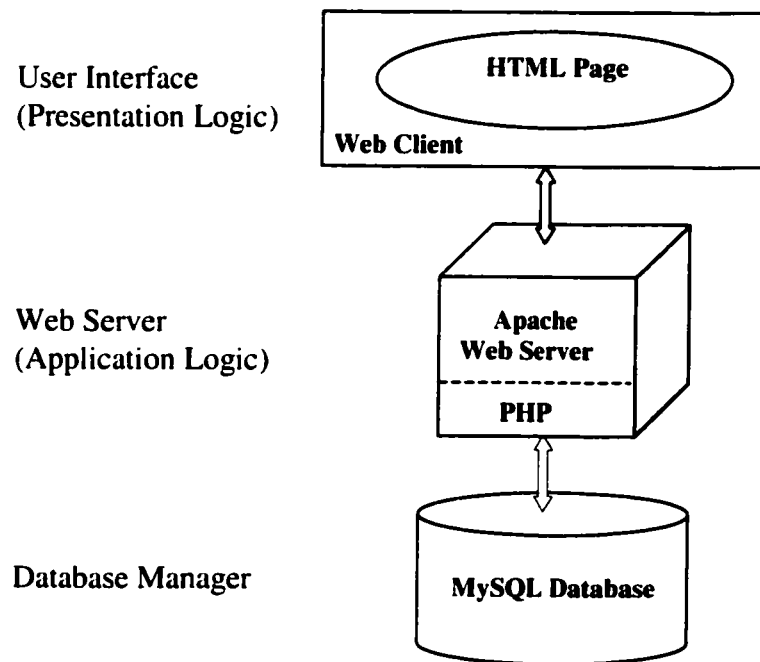


Figure 2.1.3: Three tier CINDI system architecture

2.2 MYSQL Database

2.2.1 Introduction

A database is a structured collection of data. To add, access, and process data stored in a computer database, a database management system is necessary. Since computers are very efficient at handling large amounts of data, database management plays a central role in computing, as stand-alone utilities, or as parts of other applications. MySQL is the most popular, free, open-source relational database management system (RDBMS) for the Unix/Linux platform. It is developed and provided by MySQL AB company [5].

MySQL was originally developed to handle very large databases, much faster than existing solutions and has been successfully used in highly demanding production environments for several years [5]. Through under constant development, today MySQL offers a rich and very useful set of functions. The connectivity, speed, and security make MySQL highly suited for accessing databases on the Internet. MySQL Version 3.22 table has a maximum size of about 4G. MySQL Server with some databases can contain 50,000,000 records, with 60,000 tables and about 5,000,000,000 rows.

MySQL follows ANSI SQL92 standard and includes some extension. It is aiming toward supporting the full ANSI SQL99 standard without sacrificing speed. The SQL stands for "Structured Query Language", the most common standardized language used to access databases.

2.2.2 MySQL Features

MySQL is fully multi-threaded using kernel threads, which means it can easily use multiple CPUs if available. It is developed not only to support multiple platforms such as Linux, Windows, FreeBSD, Solaris, HP-Unix etc., but also to provide multiple APIs support for C, C++, Java, Perl, PHP and Python. MySQL server controls access to the data to ensure that multiple users can work with it concurrently, to provide fast access to it, and to guarantee that only authorized users can obtain access. Its privilege and password system is very flexible and secure, and allows host-based verification. Passwords are secure because all password traffic is encrypted when connected with a server.

MySQL is very fast, reliable, and easy to use. In order to improve system performance, SQL functions are implemented through a highly optimized class library and should be as fast as possible. Usually there is no memory allocation at all after query initialization. MySQL also provides fast join operations using an optimized one-sweep multi-join, very fast B-tree disk tables with index compression and in-memory hash tables that are used as temporary tables. It has been tested with Purify, a commercial memory leakage detector, to eliminate memory leaks [5].

The performance comparison with other database systems, was carried out on the same NT 4.0 machine with MySQL benchmark suite [6]. Since the benchmark is single threaded, it can measure the minimum time for the operations. Table 2 and 3 show that MySQL is much faster.

Database	Seconds
Mysql	367
Mysql_odbc	464
Db2_odbc	1206
Informix_odbc	121126
Ms-sql_odbc	1634
Oracle_odbc	20800
Solid_odbc	877
Sybase_odbc	17614

Table 2: Comparison: Reading 2,000,000 rows by index

Database	Seconds
Mysql	381
Mysql_odbc	619
Db2_odbc	3460
Informix_odbc	2692
Ms_sql_odbc	4012
Oracle_odbc	11291
Solid_odbc	1801
Sybase_odbc	4802

Table 3: Comparison: Inserting (350,768) rows

Compared with other commercial databases such as Oracle and DB2, MySQL does not support view, cursor, trigger and stored procedure. However, most of these limitations are not considered to be very important, because MySQL Server is mostly used in applications and on web systems where the application programmer has full control on the database usage.

Current MySQL Version provides three basic table formats (ISAM, HEAP and MyISAM) and two additional table types (InnoDB, or BDB), depending on how you compile it. When you create a new table, you can tell MySQL which table type it should use for the table. In order to check referential integrity (foreign key constraints), we

choose InnoDB table as our database table type. Moreover, the InnoDB transactional table handler also offers ACID support, row-level locking, crash recovery, and multiversioning. A transaction defines a sequence of server operations that is guaranteed by the server to be atomic in the presence of multiple clients and server crashes. The acronym ACID is used to refer to the following four important properties of transactions:

- **Atomicity:** a transaction either completes successfully, and the effects of all its operations are recorded in the objects, or (if it fails or is deliberately abort) it has no effect at all.
- **Consistency:** a transaction takes the system from one consistent state to another consistent state.
- **Isolation:** each transaction must be performed without interference from other transaction; in other words, the intermediate effects of a transaction must not be visible to other transactions.
- **Durability:** after a transaction has completed successfully, all its effects are saved in permanent storage [9].

2.2.3 InnoDB Tables

Technically, InnoDB is a complete database backend placed under MySQL. InnoDB has its own buffer pool for caching data and indexes in main memory. It stores its tables and indexes in a tablespace, which may consist of several files. This is different from MyISAM tables where each table is stored as a separate file.

InnoDB provides MySQL with a transaction-safe (ACID compliant) table handler with commit, rollback, and crash recovery capabilities [5]. To increase multi-user

concurrency and performance, InnoDB transaction model has been attempting to combine the best properties of a multiversioning database with the traditional two-phase locking. InnoDB does locking on row level and provides an Oracle-style consistent non-locking read in SELECTs. The lock table in InnoDB is stored so space-efficiently that lock escalation is not needed: typically several users are allowed to lock every row in the database, or any random subset of the rows, without running out of memory. InnoDB tables support FOREIGN KEY constraints as the first table type in MySQL.

In InnoDB all user activity happens inside transactions. If the auto-commit mode is used, each SQL statement will form a single transaction. If the auto-commit mode is switched off, then a user always has a transaction open. After the SQL COMMIT or ROLLBACK statement, the current transaction ends and a new one starts. Both statements will release all InnoDB locks that were set during the current transaction. A COMMIT means that the changes made in the current transaction are made permanent and become visible to other users. A ROLLBACK on the other hand cancels all modifications made by the current transaction.

2.3 Sub-System Design

A sub-system is usually identified by the services it provides. A service is a group of related functions that share some common purpose. So a sub-system is a package of interrelated functions, operations, events and constraints, with well-defined interface to interact with other sub-systems. The system architecture design captures the high-level bird's eye view of the system, divides it into fairly independent subsystems, and specifies

the organization topology of these subsystems. Following a top-down approach, the web-based CINDI system can be separated into the following six sub-systems:

Semantic Header Database sub-system (SHDB): To store all information such as Semantic Header, authors and users information, annotation etc.

Security Control sub-system (SC): To register contributors' and users' personal information, and to check the access authorization so that only registered contributors/users can enter the CINDI system.

Resource Registration sub-system (RR): Authors/providers can create, modify and register the resource semantic headers into the SHDB system. It also contains the ASHG sub-system, which can automatically generate the draft version metadata.

Automatic Semantic Header Generator sub-system (ASHG): To generate Semantic Header for HTML, Latex, Text and RTF documents. It can be considered as part of the Resource Registration (RR) sub-system.

Search sub-system (SR): To interact with the SHDB system by entering a search query and retrieving the search results through the graphical user interface.

Annotation sub-system (AT): To display the existing annotations of the resource and allow the registered user to make new comments.

After successfully registering his personal information through the Security Control sub-system (SC), each registrant will get his account and password by e-mail. Resource Registration sub-system (RR) can accept semantic header metadata manually or

automatically. Resource contributor enters the semantic header fields such as title, alt-title, author information, keywords, subject, abstract, annotation etc. by hand. To make sure that the entered fields are acceptable, RR will check the metadata according to the rules of the semantic header before storing it into Semantic Header database (SHDB). On the other hand, the ASHG sub-system may be used to automatically generate a draft semantic header metadata for the document and asks the resource contributor to validate it. A search query is entered through the user interface of the Search sub-system (SR), which is responsible for finding the requested materials and displaying the metadata and actual resource to the user. Reviewer's comments on a specific resource can be made through the Annotation sub-system (AT)

Chapter 3

Semantic Header Database System

3.1 Semantic Header Structure

A standardized index structure was proposed by Desai [3] and is called Semantic Header. It contains those elements that are most often used in the search for an information resource such as the title, author(s), keywords, subject and sub-subject field. The structure of the Semantic Header follows extended BNF rules: tags are given in quotes, optional items are bracketed by square brackets “[” and “]”, alternative items are separated by a bar “|”, and the superscript plus sign “+” means that items may be repeated one or more times. The grammar of the Semantic Header is:

```
semantic_header := "<semantichdr> contents " </semantichdr>"
contents       := title al-title authors publisher keywords version
                source language subjects identifier coverage
                classification system_requirement genre dates
                abstract annotation
title          := "<title>" ..."</title>"
alt-title     := "<alttitle>" [...] "</alttitle>"
authors       := "<author>"author_info"</author>"
author_info   := role name organization address phone email
role          := "<role>" Author|Co_author|Editor|Artist|Designer|
                Programmer"</role>"
name         := "<name>" [...] "</name>"
```

```

organization      := "<organization>" [...] "</organization>"
address           := "<address>" [...] "</address>"
phone             := "<phone>" [...] "</phone>"
email             := "<email>" [...] "</email>"
keywords          := "<keywords>" [...] "</keywords>"
version           := "<version>" [...] "</version>"
source            := "<source>" [...] "</source>"
language          := "<language>" [Arabic|Chinese|English|French|
                        German...] "</language>"
subjects          := "<subject>" subject_group* "</subject>"
subject_group     := general sub_subject sub_sub_subject
general           := "<general>" ... "</general>"
sub_subject       := "<sub_subject>" [...] "</sub_subject>"
sub_sub_subject   := "<sub_sub_subject>" [...] "</sub_sub_subject>"
coverage          := "<coverage>" coverage_group "</coverage>"
coverage_group    := coverage_type coverage_value
coverage_type     := "<coverage_type>" Geographical|Spatial|Temporal|
                        Epoch "</coverage_type>"
coverage_value    := "<coverage_value>" [...] "</coverage_value>"
identifier         := "<identifier>" identifier_group "</identifier>"
identifier_group  := identifier_type identifier_value
identifier_type   := "<identifier_type>" FTP|ISBN|ISSN|Gopher|
                        HTTP|URN|SHN|Call No. "</identifier_type>"
Identifier_value  := "<identifier_value>" [...] "</identifier_value>"
classification    := "<classification>" classification_group
                        "</classification>"
classification_group := classification_type classification_value
classification_type := "<classification_type>" Legal|
                        Security Level "</classification_type>"
classification_value := "<classification_value>" [...]
                        "</classification_value>"
system_req        := "<system_req>" system_req_group "</system_req>"
system_req_group  := system_req_type system_req_value
system_req_type   := "<system_req_type>" Network|Software|Hardware
                        "</system_requ_type>"
System_req_value  := "<system_req_value>" [...] "</system_req_value>"
genre             := "<genre>" file_format file_size "</genre>"
file_format       := "<file_format>" ... "</file_format>"

```



```

file_size      := "<file_size>" ... "</file_size>"
dates          := "<dates>"create_date expiry_date upload_date
               "</dates>"

create_date    := "<create_date>" [ ... ] "</create_date>"
expiry_date    := "<expiry_date>" [ ... ] "</expiry_date>"
upload_date    := "<upload_date>" ... "</upload_date>"
abstract       := "<abstract>" ... "</abstract>"
annotation     := "<annotation>" [ ... ] "</annotation>"

```

3.2 Entity-Relationship Data Model

The entity-relationship (ER) data model is based on a perception of a real world that consist of a set of basic objects called entities, and of relationships among these objects. It is intended primarily to facilitate database design by allowing the specification of an enterprise schema. An important task in database modeling is to specify how entities and relationships are distinguished. The ER diagram shows the entities within the database, the associations (or relationships) among the entities, and the attributes or properties of the entities and their relationships.

Figure 3.2 shows the ER diagram of Semantic Header database. According to the structure of the Semantic Header, we can identify ten entities: resource, coverage, identifier, system_req, classification, language, subject, author, users and contributor. In the CINDI system, one resource can have more than one coverage information, but one coverage information can only be mapped to one particular resource, therefore there is a many-to-one relationship between the coverage entity and the resource entity. The same relationship exists between the identifier entity and the resource entity, between the system_req entity and the resource entity, between the classification entity and the resource entity.

There is a many-to-many relationship between the subject entity and the resource entity, because one resource may belong to many different subjects, meanwhile, one subject may belong to many different resources. One resource may have many authors, co-authors, users can access the resources and make annotations, so the many-to-many relationship can be found between the author entity and the resource entity, between the users entity and the resource entity. If one resource has different language versions, it will have different semantic headers and be considered as different resources. Hence, there is a one-to-many relationship between language entity and the resource entity.

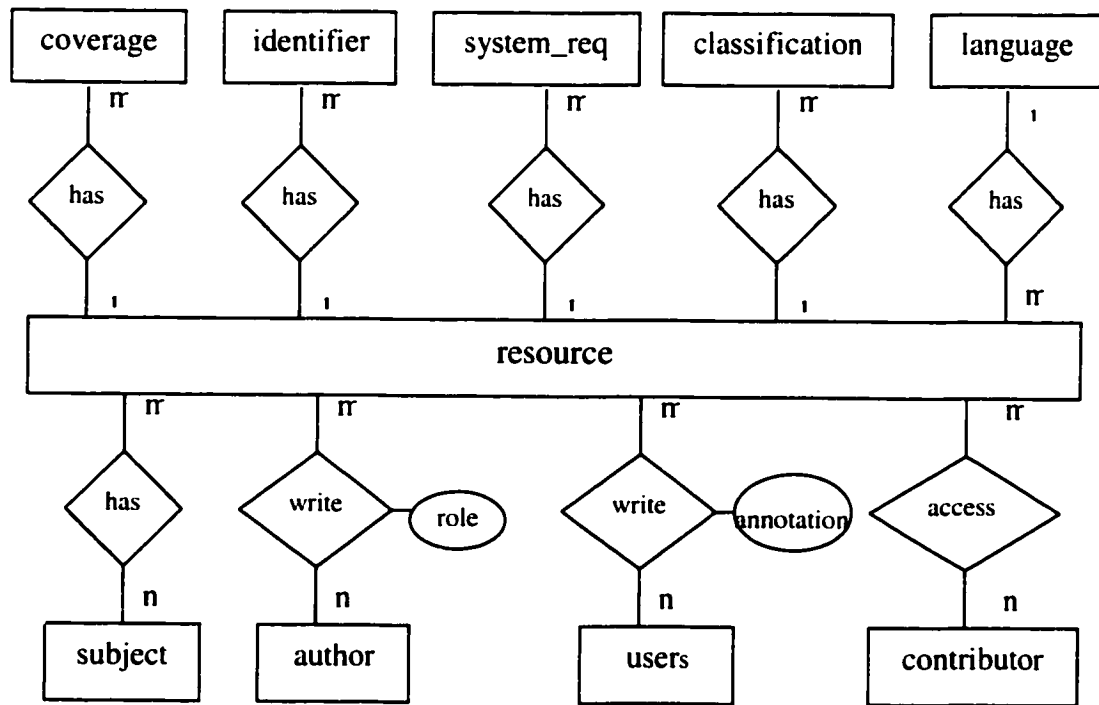


Figure 3.2: ER diagram of the Semantic Header Database

3.3 Database Table Description

The ER model is convenient for representing an initial, high-level database design. To translate an ER diagram into a collection of tables with associated constraints, i.e. a relational database schema, entity sets are mapped to relations (each attribute of the entity set becomes an attribute of the table). Relationship sets are also mapped to relations with considering the key constraints and participation constraints [10]. The goal is to make the relation schema satisfy the DK/NF normal form, which will enforce all general constraints from knowledge of the domains of the attributes and the key constraints [9]. The following sections will briefly describe the 13 tables used in the CINDI system.

3.3.1 Resource Table

The following SQL statement defines a resource table that captures the main semantic header information:

```
create table resource (  
    resource_id      int(10) unsigned NOT NULL AUTO_INCREMENT,  
    title            varchar(255) NOT NULL,  
    alt_title        varchar(255),  
    language_id      int(10) unsigned,  
    keyword           varchar(255),  
    publisher         varchar(255),  
    created_date     date NOT NULL,  
    upload_date      date NOT NULL,  
    expiry_date      date,  
    last_update      date,  
    version           varchar(50),  
    source            varchar(255),  
    size              int(50) unsigned,  
    resource_format  varchar(128),  
    abstract          text NOT NULL,  
    filename          varchar(255) NOT NULL,
```

```

hit_number      int(10) unsigned NOT NULL DEFAULT 0,
cost            int(10) unsigned DEFAULT 0,
INDEX language_ind (language_id),
primary key(resource_id),
FOREIGN KEY (language_id) REFERENCES language(language_id))
TYPE=INNODB;

```

Resource_id: primary key, it distinguishes the different resources.

Title: a required field, it is a name given to the resource by its creator or a short description of the resource.

Alt_title: an option field, it is an secondary title or a alternate short description of the resource.

Keyword: the field is used for keyword resources searching. Keywords are separated by comma.

Language_id: an option field, the language of the resource.

Publisher: an option field, it is the name who publish the resource.

Create_date, Expiry_date, Upload_date, Last_update: the create_date and the upload_date are required fields. The expiry_date and the last_update are optional fields. The upload_date and the last_update are generated by system.

Version: an option field, the version number.

Source: an option field, it contains the relationships, domains and identifiers of the related resources.

Size: an option field, it measures the size of the resource by byte.

Resource_format: an optional field, it describes the physical or electronic format of the resource such as TEXT, PDF, Postscript, GIF.

Abstract: a required text field, it is used for keyword resource searching.

Filename: a required field, it is the file name of the real resource.

Hit_number: a required field, it indicates how many times the resource is visited.

Cost: the fee of accessing the resource is given.

Here we remove the annotation field from the original version, because it is the attribute of the relationship between users entity and resource entity.

3.3.2 Subject Table

The subject table can be defined with subject ID and subject name. The subject_name field is the words or phrases indicative of the information content.

```
create table subject (  
    subject_id      int(10) unsigned NOT NULL,  
    subject_name    varchar(100) NOT NULL,  
    primary key(subject_id)  TYPE=INNODB;
```

3.3.3 Resource_subject Table

The subject of one resource consists of three sublevels: subject, sub-subject, sub-sub-subject. For example, one resource may belong to Computer Science, Information storage and retrieval, Web-based online information services. It may also belong to Computer Science, Information storage and retrieval, Online information services. The resource_id is the foreign key referencing the resource table; subject, sub_subject, sub_sub_subject are foreign keys referencing the subject table. The resource_subject table can be defined using the following SQL statement:

```
create table resource_subject (  
    resource_id     int(10) unsigned NOT NULL,  
    subject         int(10) unsigned NOT NULL,  
    sub_sub_subject int(10) unsigned NOT NULL,
```

```

sub_sub_subject  int(10)  unsigned NOT NULL,
INDEX i_reource_subject_rid  (resource_id),
INDEX  i_reource_subject_sid (subject),
INDEX  i_reource_subject_sub (sub_subject),
INDEX  i_reource_subject_sub_sub (sub_sub_subject),
primary key(resource_id, subject, sub_subject, sub_sub_subject),
FOREIGN KEY (subject) REFERENCES subject(subject_id),
FOREIGN KEY (sub_subject) REFERENCES subject(subject_id),
FOREIGN KEY (sub_sub_subject) REFERENCES subject(subject_id),
FOREIGN KEY (resource_id) REFERENCES resource(resource_id)
TYPE=INNODB;

```

3.3.4 Author Table

The SQL statement creates all the information about the author(s) of the resources. Author table includes name, organization, address, phone number and e-mail address fields. Here we remove the role field from the author table to the resource_author table, because it is the attribute of the relationship between author entity and resource entity. The part of the name field is used to create the column index, which will significantly improve the resource searching performance.

```

create table author (
    author_id      int(10) unsigned NOT NULL AUTO_INCREMENT,
    name           varchar(100) NOT NULL,
    organization    varchar(255),
    address        varchar(255),
    phone          varchar(50),
    e_mail         varchar(50),
    apt_no         varchar(20),
    city           varchar(30),
    province       varchar(30),
    country        varchar(30),
    p_code         varchar(20),
    primary key(author_id)) TYPE=INNODB;

```

```
Create index i_author_name on author(name(15));
```

3.3.5 Resource_author Table

We can map the many-to-many relationship between the resource entity and the author entity into a resource_author table. Typical values for the role field could be author, co-author, designer, editor, programmer, creator, artist, publisher, etc. Assume that one author can only play one role in the same resource, we define its primary key as the combination of resource_id and author_id, both of them are also the foreign keys referencing the resource table and author table to maintain the integrity constraint.

```
create table resource_author (  
    resource_id      int(10) unsigned NOT NULL,  
    author_id        int(10) unsigned NOT NULL,  
    role             varchar(20),  
    INDEX i_resource_author_rid (resource_id),  
    INDEX i_resource_author_aid (author_id),  
    primary key (resource_id, author_id),  
    FOREIGN KEY (resource_id) REFERENCES resource(resource_id),  
    FOREIGN KEY (author_id) REFERENCES author(author_id))  
TYPE=INNODB;
```

3.3.6 Users Table

The users table is used to authenticate the access privilege and collect the personal information such as first name, last name, address, phone number and e-mail address. The security control sub-system will check the username and password to make sure that only registered user can access the resource searching system. The user also can make annotations for a set of resources.

```
create table users (  

```

```

user_id          varchar(20) NOT NULL,
passwd          varchar(20) NOT NULL,
fname           varchar(50) NOT NULL,
lname           varchar(50) NOT NULL,
address         varchar(50),
apt_no          varchar(50),
city            varchar(30),
province        varchar(30),
country         varchar(30),
p_code         varchar(20),
e_mail         varchar(50),
primary key(user_id) TYPE=INNODB;

```

3.3.7 Annotation Table

The following SQL statement defines the annotation table, which is corresponding to the many-to-many relationship between the resource entity and the users entity. Once a user inputs an annotation, he cannot modify it, but he/she can make any number of comments on the same resource. Both the resource_id and user_id are the foreign keys referencing the resource table and users table.

```

create table annotation (
    resource_id    int(10)    unsigned NOT NULL,
    user_id        varchar(20) NOT NULL,
    annotation     text,
    INDEX i_annotation_rid (resource_id),
    INDEX i_annotation_uid (user_id),
    FOREIGN KEY (resource_id) REFERENCES resource(resource_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id)
    TYPE=INNODB;

```

3.3.8 Contributor Table

Similar to the users table, the contributor table contains the first name, last name, organization name, detailed address information, e-mail etc. Only registered contributor

can upload the resources and provide the semantic header metadata manually or automatically through the resource registration sub-system. All the available information about the contributor table is captured by the following SQL definition:

```
create table contributor (  
    contributor_id    varchar(20) NOT NULL,  
    passwd            varchar(20) NOT NULL,  
    fname             varchar(50) NOT NULL,  
    lname             varchar(50) NOT NULL,  
    organization       varchar(100),  
    department        varchar(50),  
    address            varchar(50),  
    apt_no            varchar(50),  
    city              varchar(30),  
    province          varchar(30),  
    country            varchar(30),  
    p_code            varchar(20),  
    e_mail            varchar(50),  
    primary key(contributor_id)    TYPE=INNODB;
```

3.3.9 Language Table

The language_id is stored in the resource table instead of the particular language name such as English, French. It provides a more flexible way to manipulate (sort, add, delete) the language. The SQL definition for the language table is given below.

```
create table language (  
    language_id        int(10) unsigned NOT NULL AUTO_INCREMENT,  
    language_name      varchar(20),  
    primary key(language_id)    TYPE=INNODB;
```

3.3.10 Coverage Table

The following SQL statement defines the coverage table, which includes two fields: a domain (target audience, coverage in a spatial and/or temporal sense, etc) and the corresponding value. For example, one resource may have coverage both in a spatial and also a temporal sense. The one-to-many relationship between the resource entity and the coverage entity can be mapped into the coverage table with the resource_id as the foreign key.

```
create table coverage (  
    resource_id      int(10) unsigned NOT NULL,  
    coverage_domain  varchar(50),  
    coverage_value   varchar(200),  
    INDEX i_coverage_rid (resource_id),  
    UNIQUE (resource_id, coverage_domain, coverage_value),  
    FOREIGN KEY (resource_id) REFERENCES resource(resource_id))  
    TYPE=INNODB;
```

3.3.11 System_req Table

The system requirement table contains a domain of the system requirements (possible value are: hardware, software, network, protocol, etc.) and the corresponding value. One resource may have hardware and software requirements as well as network requirements. The following SQL statement captures the one-to-many relationship between the resource entity and the system requirement entity with the resource_id as the foreign key.

```
create table system_req (  
    resource_id      int(10) unsigned NOT NULL,  
    system_req_domain  varchar(50),  
    system_req_value   varchar(200),  
    INDEX i_system_req_rid (resource_id),  
    UNIQUE (resource_id, system_req_domain, system_req_value),  
    FOREIGN KEY (resource_id) REFERENCES resource(resource_id))  
    TYPE=INNODB;
```

3.3.12 Identifier Table

The identifier table is defined with two fields: the domain and the corresponding value. The possible domain values are: ISBN, URL, ISSN, etc. One resource may have more than one identifier information, for example, one resource may be identified by a URL and a ISBN. But one identifier information can only be mapped to one particular resource. The one-to-many relationship between the resource entity and the identifier entity can be mapped into the identifier table with the resource_id as the foreign key.

```
create table identifier (  
    resource_id      int(10) unsigned NOT NULL,  
    identifier_domain varchar(50),  
    identifier_value  varchar(200),  
    INDEX i_identifier_rid (resource_id),  
    UNIQUE (resource_id, identifier_domain, identifier_value),  
    FOREIGN KEY (resource_id) REFERENCES resource(resource_id)  
    TYPE=INNODB;
```

3.3.13 Classification Table

The classification table consists of two fields: the domain and the corresponding value. The possible domain values are: legal, security level, etc. One resource may have more than one classification information, for example, one resource may have security restriction and copyright status. The following SQL statement maps the one-to-many relationship between the resource entity and the classification entity into the classification table, with the foreign key resource_id.

```
create table classification (  
    resource_id      int(10) unsigned NOT NULL,  
    clasf_domain     varchar(50),  
    clasf_value      varchar(200),
```

```
INDEX i_classification_rid (resource_id),  
UNIQUE (resource_id, clasf_domain, clasf_value),  
FOREIGN KEY (resource_id) REFERENCES resource(resource_id)  
TYPE=INNODB;
```

3.4 Primary and Foreign Keys

A primary key is the combination of the values of one or more attributes that collectively and uniquely identify an entity. To ensure entity integrity, each component of a primary key cannot accept a null value. A foreign key is the primary key of another table. To ensure referential integrity and keep data consistent, the foreign key in the referencing table must match the primary key of the referenced table.

For the resource table, primary key resource_id is added to distinguish each resource, the MySQL function auto_increment will generate a sequence number for each record. For the author table, author_id is chosen as the primary key to distinguish the same author name. If the personal information such as name, organization, address, phone number and e-mail address, etc. of two authors are exactly same, they will be treated as one author (the same author_id). Each contributor has his own personal information. The contributor_id and user_id are the primary keys of contributor table and users table separately. For the subject table, the number field subject_id is defined as the primary key to speed up the searching performance.

As shown in Table 3, the foreign keys of the resource_author table are: author_id and resource_id, which represent the relation between the author and the resource. All the authors' information of a particular resource can be obtained through resource_author

table. Through the two foreign keys `subject_id` and `resource_id` of the `resource_subject` table, all related subjects of one particular resource can be retrieved. In the annotation table, the same user (foreign key `user_id`) can make many different annotations on the same resource (foreign key `resource_id`). For the identifier table, coverage table, `system_req` table and classification table, the foreign key `resource_id` is used to link with the resource table.

Table Name	Primary Key	Foreign Key
resource	resource_id	language_id
subject	subject_id	
resource_subject	resource_id, subject_id	resource_id, subject_id
author	author_id	
resource_author	resource_id, author_id	resource_id, author_id
users	user_id	
annotation	all attributes	resource_id, user_id
contributor	contributor_id	
language	language_id	
coverage	all attributes	resource_id
system_req	all attributes	resource_id
identifier	all attributes	resource_id
classification	all attributes	resource_id

Table 4: Primary key and foreign key for each table

3.5 Intermediate Tables of ASHG Subsystem

When a contributor uploads a new resource to the CINDI system, Automatic Semantic Header Generator (ASHG) can generate the draft Semantic Header fields such as title, author, keywords, subjects, author's information, annotation, coverage, system

requirement, identifier and classification from the document. Since the Semantic Header information generated by the ASHG may not be accurate, the contributor is required to view it and modify it. After validated by the contributor, the meta-data will be stored into the Semantic Header database. This makes the index entry in the database more accurate, more useful.

In order to display and check the Semantic Header that is automatically generated by ASHG, several intermediate tables are implemented to temporarily store this information. For example, the title, keywords, created_date, abstract, annotation are put into the sh table. The author's information of the resource is stored in the sh_author table. To allow more than three subjects for each resource, the subject information is put into the sh_subject table. The other information such as coverage, identifier, system requirements and classification is separately stored into the sh_covearage table, the sh_indentifier table, the sh_system_req table and the sh_classification table.

After the contributor uploads a resource, the database system will generate a unique ID for it; this is important for a concurrent multi-user. Once ASHG stores the draft Semantic Header of the resource into the intermediate tables, this ID plus the filename will be used to identify the corresponding information for the resource. The following SQL statements implement these intermediate tables in MySQL database. Their structure is very similar to the structure of resource, author, coverage, system_req, identifier and classification tables. In the sh table, we add the annotation field, the reason is that ASHG can only generate one annotation for each uploaded document.

```
create table sh (
```

```

resource_id      int(10) unsigned NOT NULL AUTO_INCREMENT,
title           varchar(255) NOT NULL,
alt_title       varchar(255),
language        varchar(30),
keyword         varchar(255),
publisher       varchar(255),
created_date    date,
expiry_date     date,
version         varchar(50),
source          varchar(255),
annotation      text,
size           int(50) unsigned,
resource_format varchar(128),
abstract       text NOT NULL,
filename       varchar(255) NOT NULL,
cost           int(10) unsigned DEFAULT 0,
primary key(resource_id)  TYPE=INNODB;

```

```

create table sh_author (
    resource_id      int(10) unsigned NOT NULL,
    name            varchar(100) NOT NULL,
    organization     varchar(255),
    address         varchar(255),
    phone           varchar(50),
    e_mail          varchar(50),
    apt_no          varchar(20),
    city            varchar(30),
    province        varchar(30),
    country         varchar(30),
    p_code          varchar(20),
    role            varchar(20) NOT NULL,
    INDEX i_sh_author_rid (resource_id),
    FOREIGN KEY (resource_id) REFERENCES sh(resource_id))
TYPE=INNODB;

```

```

create table sh_subject (
    resource_id      int(10) unsigned NOT NULL,

```

```

subject          varchar(100) NOT NULL,
sub_subject      varchar(100) NOT NULL,
sub_sub_subject  varchar(100) NOT NULL,
INDEX i_sh_subject_rid (resource_id),
primary key(resource_id, subject, sub_subject,
             sub_sub_subject),
FOREIGN KEY (resource_id) REFERENCES sh(resource_id)
TYPE=INNODB;

```

```

create table sh_coverage (
resource_id      int(10) unsigned NOT NULL,
coverage_domain  varchar(50),
coverage_value   varchar(200),
INDEX sh_coverage_rid (resource_id),
UNIQUE (resource_id, coverage_domain, coverage_value),
FOREIGN KEY (resource_id) REFERENCES sh(resource_id)
TYPE=INNODB;

```

```

create table sh_system_req (
resource_id      int(10) unsigned NOT NULL,
system_req_domain varchar(50),
system_req_value varchar(200),
INDEX sh_system_req_rid (resource_id),
UNIQUE (resource_id, system_req_domain, system_req_value),
FOREIGN KEY (resource_id) REFERENCES sh(resource_id)
TYPE=INNODB;

```

```

create table sh_identifier (
resource_id      int(10) unsigned NOT NULL,
identifier_domain varchar(50),
identifier_value  varchar(200),
INDEX sh_identifier_rid (resource_id),
UNIQUE (resource_id, identifier_domain, identifier_value),
FOREIGN KEY (resource_id) REFERENCES sh(resource_id)
TYPE=INNODB;

```

```

create table sh_classification (
resource_id      int(10) unsigned NOT NULL,

```



```
clasf_domain      varchar(50),
clasf_value       varchar(200),
INDEX sh_classification_rid (resource_id),
UNIQUE (resource_id, clasf_domain, clasf_value),
FOREIGN KEY (resource_id) REFERENCES sh(resource_id))
TYPE=INNODB;
```

3.6 Tuning Performance

To retrieve accurate data in as little time as possible is the goal of the CINDI system. Several factors should be considered to improve the database performance, for example, indexing of tables, streamlining SQL statements, and concurrent user transactions.

3.6.1 Using Indexes

Data can be retrieved from a database using two methods. The first method, often called Sequential Access Method, requires SQL to go through each record looking for a match. This search method is inefficient, but it is the only way for SQL to locate the correct record. Adding indexes to the database enables SQL to use the Direct Access Method, it can quickly find the right position of the data file without having to check every data record. Indexes can improve the speed of data retrieval, but they will slow data updates and take up space within the database. In order to increase the speed of join in resource searching, we always index on fields that are used in joins between tables such as resource_id, author_id, subject_id, user_id, language_id. We also build an index on the part of the name field in the author table, to facilitate the query based on author name criteria. Because most names usually differ in the first 15 characters, it almost achieves the same performance as indexing the whole name field [5]. Also, using partial

columns for indexes can make the index file much smaller, which could save a lot of disk space.

3.6.2 Streamline SQL Statement

Streamlining SQL is the process of finding the optimal arrangement of the elements within a query, particular in the **WHERE** clause. The arrangement of conditions depends on the columns that are indexed, as well as on which condition will receive the fewest records. The objective is to narrow down the results of the SQL statement by using an indexed column that returns the fewest number of rows. The condition that returns the fewest records in a table is said to be the most restrictive condition.

When the query optimizer reads the most restrictive condition first, it is able to narrow down the first set of results before proceeding to the next condition. The next condition, instead of looking the whole table, should look at the subset that was selected by the most selective condition. Ultimately, data is retrieved faster. In MySQL, we should place the most restrictive condition last in the **WHERE** clause, but it depends on the order of the processing steps in a specific implementation. For example, Oracles's query optimizer reads a **WHERE** clause from the bottom up, so in a sense, we should place the restrictive condition first.

In the CINDI system, for the worst case, five tables: resource, subject, resource_subject, author, resource_author, are needed to join together to perform a resource search query on Semantic Header database. MySQL resolves all joins using a single-sweep multi-join method. This means that MySQL reads a row from the first

table, then finds a matching row in the second table, then in the third table and so on. When all tables are processed, it outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table. If the large tables are joined first, the size of the tables will increase significantly and negatively impact on the successive joins. To achieve better performance, we arrange the join order as author, resource_author, subject, resource_subject and resource (see section 6.2 for details).

3.6.3 Using Persistent Connection

Each time, a connection between the web server and the MySQL database must be established in order to store or retrieve data. There is a way to reduce the connection overhead, i.e. we can use the PHP function `mysql_pconnect()` to setup a persistent connection with the MySQL server. When a persistent connection is requested, the system will check whether or not there's already an identical persistent connection (a connection that was opened to the same host before, with the same username and the same password, without calling the `mysql_close($db)` function). If the connection exists, PHP will use it instead of creating a new one. It is more efficient than non-persistent connection and achieves better performance [8].

3.6.4 Minimize Transaction

As we discussed in section 2.2.2, a transaction defines a logical unit of operations that either wholly succeed or has no effect on the database state. Transactions are usually controlled by the database management system that must adhere to the ACID properties,

which also ensure that unfinished updates or corrupting activities are not committed to the database. The transactional paradigm has its benefits and its drawbacks. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be, or is necessary. We can combine many statements and accept these all in one go with the COMMIT command. If an update fails, all the changes will be restored (before the COMMIT, all changes that have taken place are not permanent). However, it also introduces lots of overhead, takes more disk space and memory to do updates. Non-transactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In the web-based CINDI system, many contributors may concurrently insert their semantic header meta-data into the database (see details in section 5.2). A transaction must be performed to make sure they are atomic operations and do not interfere with each other. The following PHP/MySQL code shows this transaction, which consists of eight insert statements:

```
BEGIN;  
$result = mysql_query("insert into resource values(...)");  
$result0 = mysql_query("insert into annotation values(...)");  
$result1 = mysql_query("insert into identifier values(...)");  
$result2 = mysql_query("insert into coverage values(...)");  
$result3 = mysql_query("insert into system_req values(...)");  
$result4 = mysql_query("insert into classification values(...)");  
$result5 = mysql_query("insert into resource_subject values(...)");  
$result6 = mysql_query("insert into resource_author values(...)");  
COMMIT;
```

In order to optimize the performance and reduce the transaction overhead, we will try to minimize the transaction process. For example, the following revised PHP/MySQL code only uses two SQL statements to substitute the above insert transaction.

```

BEGIN;
//insert record into table resource
$result = mysql_query("insert into resource values
    (null,'title', null, 3, null, null, '0000-01-01','0000-01-01',
    '0000-01-01','0000-01-01', null, null, 10, null, 'abstract',
    'filename', 0, 0)");

$paperid = mysql_insert_id($db);
COMMIT;

$result11 = mysql_query("UPDATE resource SET title='$title',
    alt_title='$alt_title', language_id=$language,
    keyword='$keyword', publisher='$publisher',
    created_date='$publish_date', upload_date='$date',
    expiry_date='$expiry_date', last_update='$date',
    version='$version', source='$source', size=$size,
    resource_format='$type', abstract='$abstract',
    filename='$f_name' WHERE resource_id=$paperid" );

// insert record into table annotaion
if ($annotation != "" )
    $result0 = mysql_query("insert into annotation values
        ($paperid,'xm','$annotation')");

// insert record into table identifier
if ($identifier != "" )
    $result1 = mysql_query("insert into identifier values
        ($paperid,'$identifier_type','$identifier')");

// insert record into table coverage
if ($coverage != "" )
    $result2 = mysql_query("insert into coverage values
        ($paperid,'$coverage_type','$coverage')");

// insert record into table system_req
if ($sysrequirement != "" )
    $result3 = mysql_query("insert into system_req values
        ($paperid,'$sysrequirement_type','$sysrequirement')");

// insert record into table classification
if ($classification != "" )
    $result4 = mysql_query("insert into classification values
        ($paperid,'$classification_type','$classification')");

// insert record into table resource_subject
if ($subject1 != 0 )
    $result5 = mysql_query("insert into resource_subject values
        ($paperid,$subject1,$sub_subject1,$sub_sub_subject1)");

if ($subject2 != 0 )
    $result6 = mysql_query("insert into resource_subject values
        ($paperid,$subject2,$sub_subject2,$sub_sub_subject2)");

if ($subject3 != 0 )
    $result7 = mysql_query("insert into resource_subject values
        ($paperid,$subject3,$sub_subject3,$sub_sub_subject3)");

```

```
.  
.   
.   
// if all above operations are successful, insert into resource_author  
$result = mysql_query("insert into resource_author values  
    ('$paperid', '$people_id', '$role')");
```

Here we successfully manage the unique identifier *resource_id* to ensure concurrent usage of the system safe by defining it as an AUTO_INCREMENT column and calling either the SQL function LAST_INSERT_ID() or the C API function mysql_insert_id(), because the last generated *resource_id* is maintained in the server on a per-connection basis, it will not be changed by another client. Based on the foreign key resource_id, the corresponding resource information can be stored into the resource_subject, resource_author, annotation, identifier, coverage, classification, system_req tables. If there is a failure in the middle of these insertions, no record will be inserted into the resource_author table for this resource. As a result, the resource search sub-system will consider it as an incomplete resource and will delete it from the database later.

Chapter 4

Security Control Sub-system

4.1 Apache SSL Server

Apache-SSL is a secure Web server, based on Apache and SSLeay/OpenSSL [11]. The Secure sockets Layer (SSL) protocol was designed to facilitate secure communication between web server and browser. SSL can be adapted as an add-on module, so we can just add the SSL module to the Apache server to improve the security without changing one single line of code. SSL not only encrypts the data flowing over the Internet, but also provides the means for both parties to authenticate each other. Authentication is the process of verifying identity so that one entity can be sure that another entity is who it claims to be.

SSL uses an encryption technique called RSA public key cryptography [12]. Public key encryption technique uses a pair of asymmetric keys for encryption and decryption. Each pair of keys consists of a public key and a private key. The public key is made public by distributing it widely. The private key is never distributed; it is always kept secret. Data that is encrypted with the public key can be decrypted only with the private key. Conversely, data encrypted with the private key can be decrypted only with the public key. The server side of the connection sends the client its public key for encrypting information; only the server can decrypt the information with the private key

it holds. The client side uses the public key to encrypt and send the server its own key, identifying it uniquely to the server and preventing onlookers at points between the two systems from mimicking either server or client.

The SSL layer exists between the transport layer and the application layer. It encrypts the data from HTTP application before passing it down to the TCP layer. When data from HTTP application are sent over the SSL layer, they are broken into several manageable packets. Each packet is compressed, and attached a message authentication code calculated using a hashing algorithm, after being encrypted, the data are combined with the header information and sent to the network.

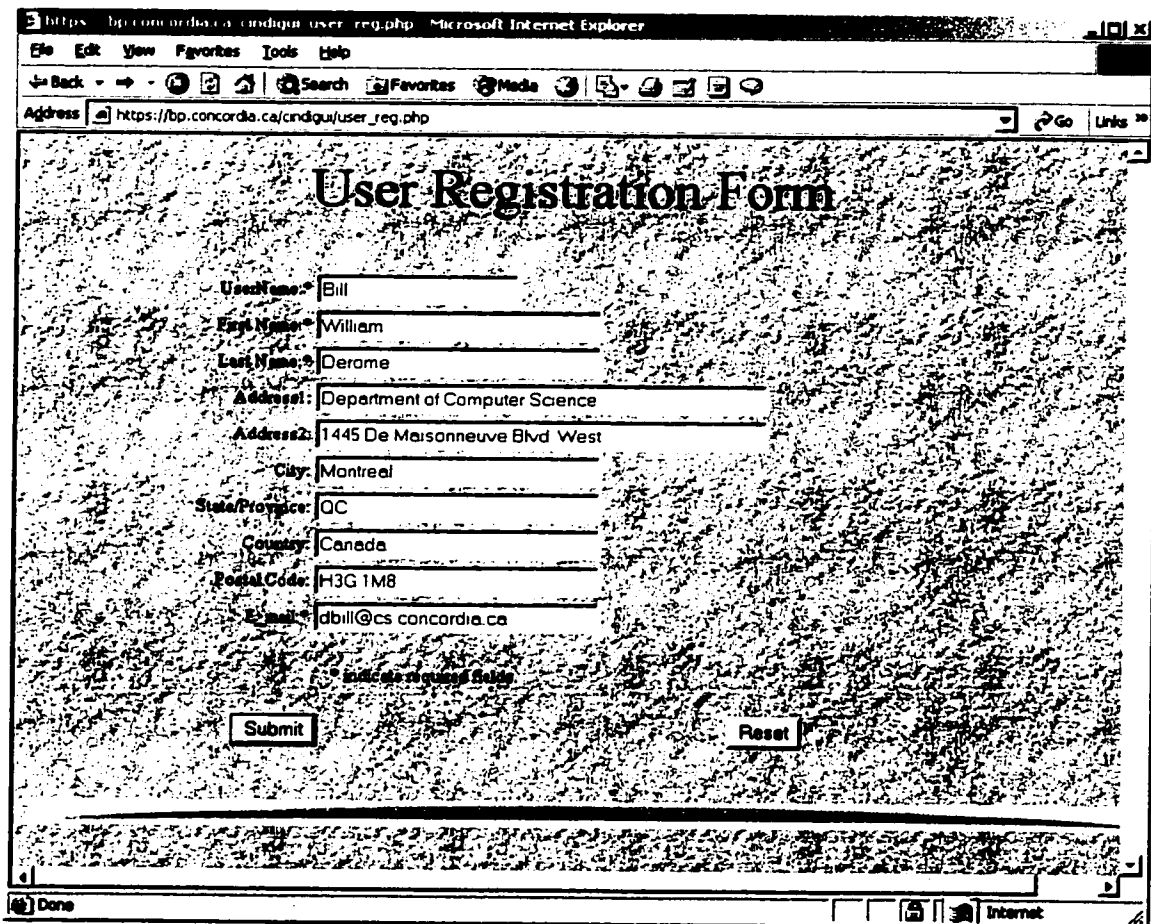
Having a secure web server is a vital necessity if you are exchanging sensitive information such as credit card number or personal information on-line. In the CINDI system, there are a couple of modules that need high security such as the user/contributor registration module and login module, in which the user inputs private information and stores it into the database.

4.2 User/Contributor Registration

The CINDI system supports two kinds of users: the normal users can query the Semantic Header database to find the useful information and make annotation; the contributors can submit the semantic header information and upload the resource into the database.

Figure 4.2.1 shows the user registration graphic interface. It asks the users to enter the personal information to register themselves. The username, first name, last name,

and email address are required fields. After the user has filled in the required information, the PHP program will connect with the backend MySQL database to check the username. If the same username already exists in our database, the system will display the error message: “User ID already exists. Please choose another one”. If the user forgets to input the required fields, the system will respond with the corresponding error messages (as shown in Figure 4.2.2). Moreover, the PHP script also checks the email address to make sure its format is correct.



The image shows a screenshot of a web browser displaying a "User Registration Form". The browser's address bar shows the URL "https://bp.concordia.ca/cndiqui/user_reg.php". The form contains the following fields and values:

Field	Value
Username	Bill
First Name	William
Last Name	Derame
Address1	Department of Computer Science
Address2	1445 De Maisonneuve Blvd West
City	Montreal
State/Province	QC
Country	Canada
Postal Code	H3G 1M8
Email	dbill@cs.concordia.ca

At the bottom of the form, there are two buttons: "Submit" and "Reset". A small asterisk and the text "Indicate required fields" are visible below the email field.

Figure 4.2.1: User registration interface

When the username and other required information are validated, the user’s personal information will be inserted into the database, the system will generate an eight

NOTE TO USERS

Page(s) not included in the original manuscript are unavailable from the author or university. The manuscript was microfilmed as received.

46

This reproduction is the best copy available.

UMI[®]

and password, clicks the submit button, the PHP script will establish a connection with the MySQL database, perform a query on the users (or contributor) table to validate the user_id (or contributor_id) and password. If the matched user_id (or contributor_id) and password are found, a non-zero number is returned and the web server will redirect the user (or contributor) to the corresponding page (for example, the search interface for users as shown in Figure 6). Otherwise, the returned zero value means that there is a mismatch for the user_id (or contributor_id) or password, the browser will display an error message: “Invalid Login. Please check your user name and password”.

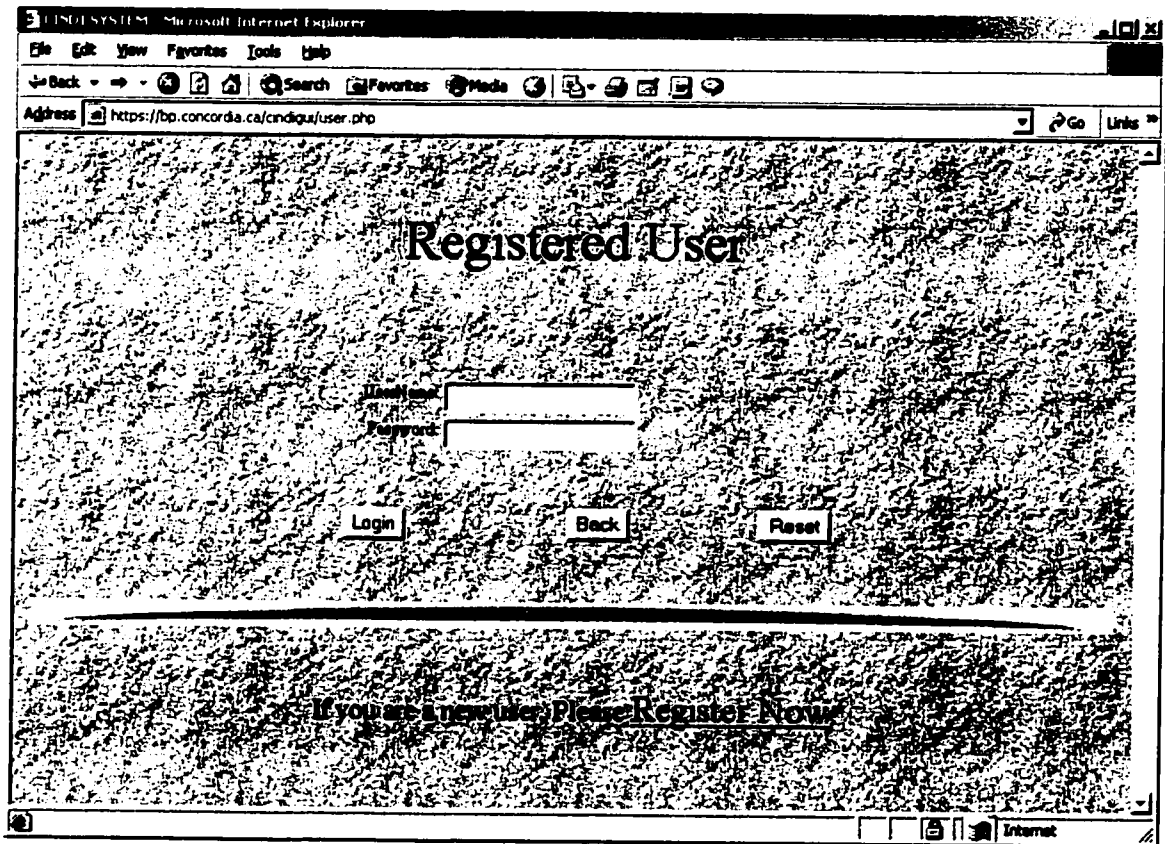


Figure 4.3: User login interface

4.4 Session Handling

There is a common security problem for web application, i.e. if a visitor knows which page he want to go, for example “register.php”, he can just type the URL address of the page such as “https://bp.concordia.ca/cindigui/register.php” in the browser, and get in that page without username and password. In this way, the user/contributor registration system seems useless. The CINDI system invokes the idea of session to solve this problem. The concept is as follows: once you register global variables with the session handler, the values of these variables are saved in files on the server. When the user requests another page, these variables are restored to the global scope. The session identifier is a long series of numbers and letters and is sent to the user as a cookie. So we can have information to associate with each user, and pass them from page to page. Each visitor accessing our system will register a unique variable (such as user_id or contributor_id) when he first logs in. The PHP program will check the unique registered variable at the beginning of each page. If this variable is found unmatched, the server will not display the correct page content (for example, Figure 6). Instead, it will show a warning message, and ask the user to login (as shown in Figure 4.4). After the user logout the system, all the registered variables will be erased.

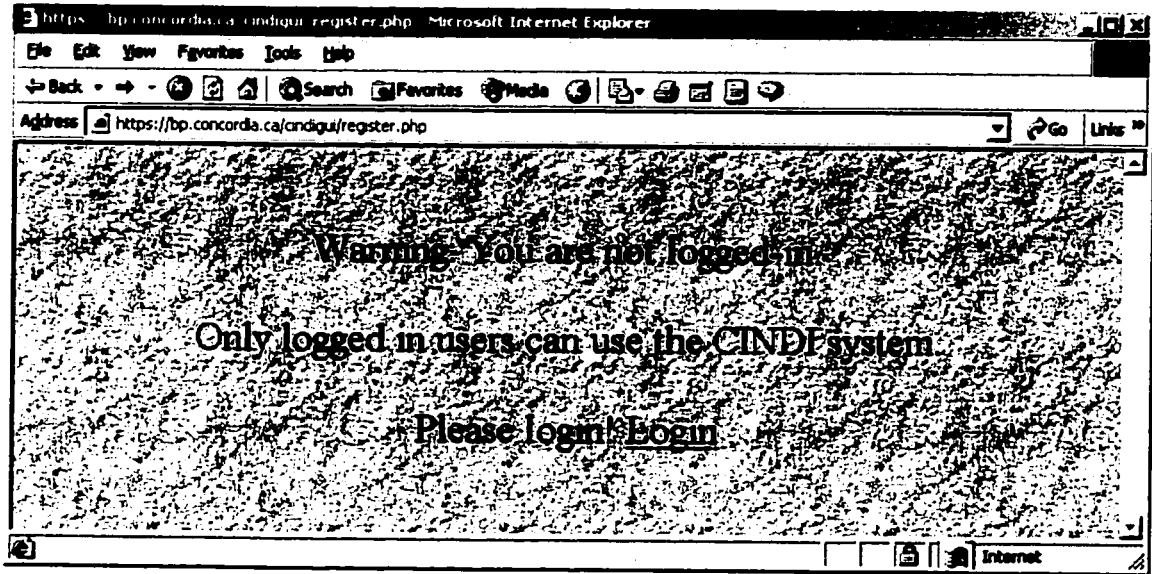


Figure 4.4: Warning page for illegal user

Chapter 5

Resource Registration Sub-system

The CINDI system provides the contributor two different ways to register resources: automatically generate a draft semantic header or manually input the semantic header. The Automatic Semantic Header Generator (ASHG) can accept HTML, TEXT, RTF, and LATEX format files; for the other formats, the contributor has to invoke the manual uploading. Once the contributor logs into the system successfully, the graphical interface for resource registration (shown in Figure 5) is presented, which divides the resource

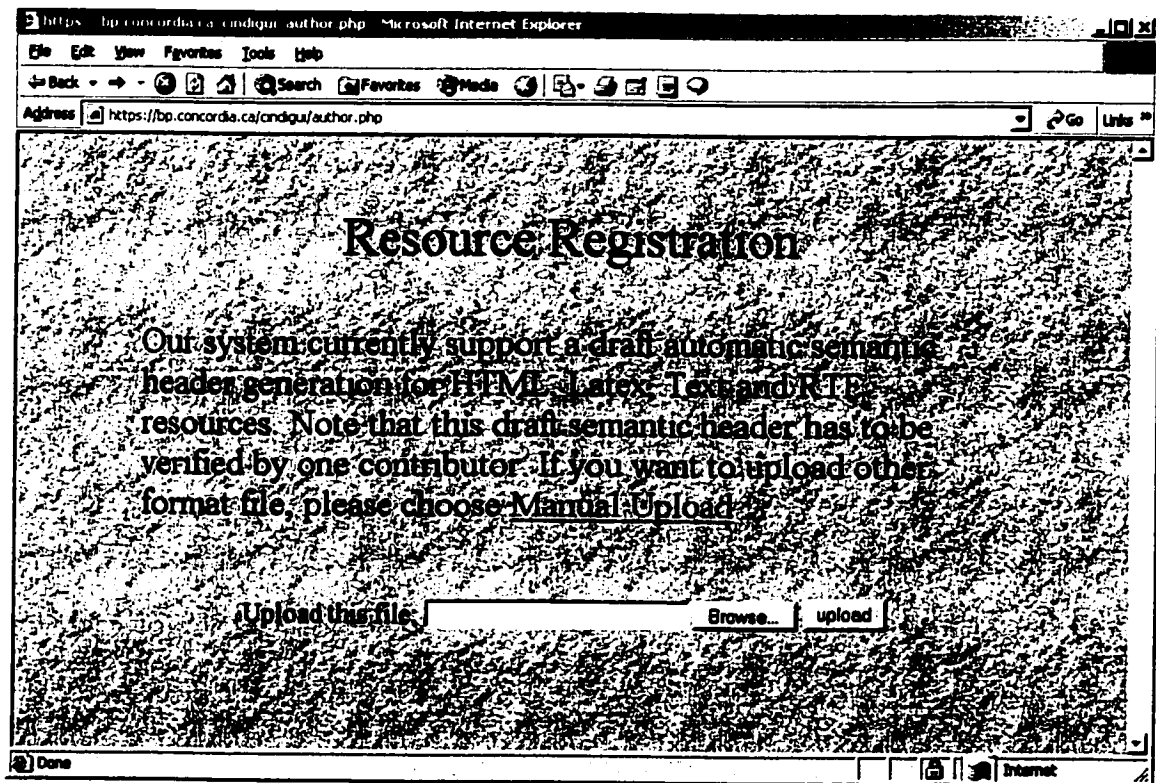


Figure 5: Resource registration sub-system interface

registration sub-system into two parts: manual resource registration, and automatic resource registration.

5.1 Manual Resource Registration

If the contributor clicks on “Manual Upload” link in Figure 5, the manual uploading method will be used to register the semantic header. As shown in Figure 5.1, the semantic header includes title, alt-title, Author, publisher, keyword, version, source, language, subject, identifier, coverage, classification, system requirement, date, abstract, annotation and filename, which are described in section 3.3. The title, author, subject, abstract and the filename are required fields (they are marked with asterisk in Figure 5.1).

After the contributor fills in the HTML form, all the information will be delivered to the web server, where the PHP program will connect with the database and insert the data into separate tables such as resource, annotation, identifier, coverage, classification, system_req, resource_subject, and resource_author. As mentioned in section 3.6.4, the whole transaction can be handled much more efficiently by defining the resource_id as an AUTO_INCREMENT column and using the SQL function LAST_INSERT_ID() or the C API function mysql_insert_id(). The AUTO_INCREMENT attribute can generate a unique resource_id for each new resource, which is maintained in the server on a per-connection basis and will not be changed by another client. Then the function LAST_INSERT_ID() or mysql_insert_id() can be used to retrieve the last automatically generated value resource_id to ensure concurrent usage of the system safe. Based on the foreign key resource_id, the corresponding resource information can be stored into the resource_subject, resource_author, identifier, coverage, classification, system_req tables.

https://bp.concordia.ca/andigu/manu_upload.php?subjectChoice1=1&sub_subjectChoice1=1027&cpTitle=A%20Natural%20language%20Proce...

Resource Manual Registration

Title: A Natural Language Processor for CINDI

All-ids: _____

Author/Other Agents: Bipin C Desai, L Kasseim, N Stratica (Use comma to separate)

Publisher: _____

Keyword: Cindi, Semantic Header, Natutal Language Processim (Use comma to separate)

Version: _____

Source: _____

Language: English

Subject:

General	Computer Science
Sub-Subject	Database management
Sub-Sub-Subject	Database applications

Subject2:

General	_____
Sub-Subject	_____
Sub-Sub-Subject	_____

Subject3:

General	_____
Sub-Subject	_____
Sub-Sub-Subject	_____

Identifier: URL www.cs.concordia.ca

Coverage: _____

Classification: _____

System Requirement: _____

Created date: 10 July 2001

Entry type: Select day | Select Month | Select Year

Abstract: In this paper, we present our work in querying a Virtual Library

Done Internet

Figure 5.1: Manual resource registration interface (Part 1)

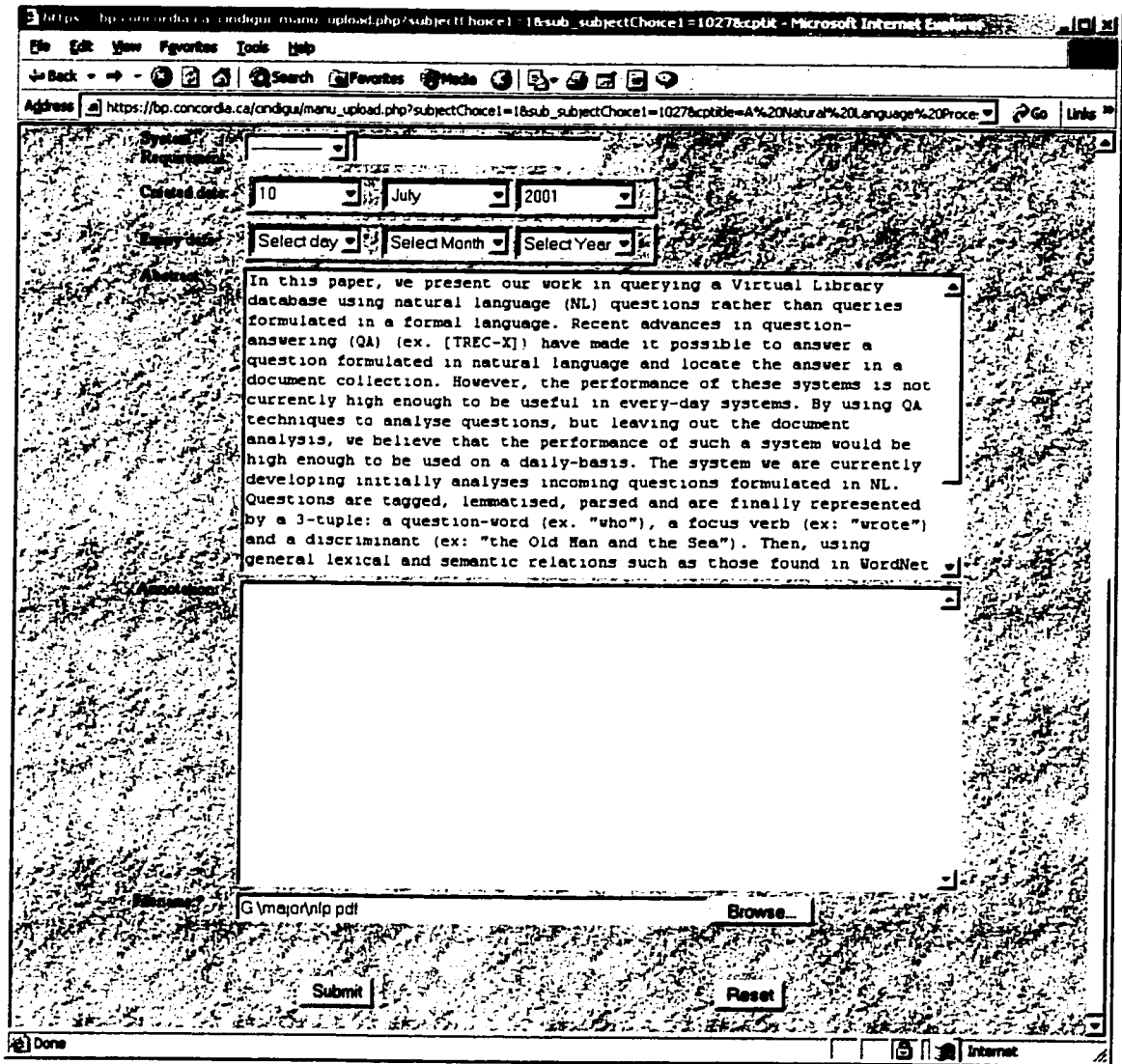


Figure 5.1: Manual resource registration interface (Part 2)

5.2 Automatic Resource Registration

Automatic Semantic Header Generator (ASHG) can generate a draft version of the semantic header for HTML, Latex, Text and RTF documents. It is developed specifically for the automatic resource registration by providing the contributor the initial semantic header meta-information, such as the document's title, abstract, keywords, subjects, dates, author's information, when he uploads a new document to the CINDI system.

The main procedure for automatic resource registration is described below:

1. Upload the selected file as indicated in the form shown in Figure 5.
2. Identify the document's format: HTML, Latex, Text and RTF.
3. Call the corresponding ASHG extractor and classify the meta-information by using frequency occurrence and positional schemes, ASHG measures the significance of the words found in the previously mentioned list.
4. Store the draft semantic header: the generated meta-information is stored into the intermediate tables such as sh, sh_author, sh_subject, sh_identifier, sh_coverage, sh_classification, sh_system_req (refer to section 3.4 for the table details).
5. Display the draft semantic header: Figure 5.2.1 shows the corresponding semantic header information by querying the intermediate tables.
6. Validate the meta-data: the contributor can verify and modify the draft semantic header generated by ASHG to guarantee its accuracy.
7. Store the modified semantic header: after clicking the submit button, the final version of meta-data is stored into the Semantic Header database (SHDB).

https://bp.concordia.ca/ondiga/auto_upload.php Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Search Favorites Media Print Mail

Address https://bp.concordia.ca/ondiga/auto_upload.php Go Links

Semantic Header of this Resource

Please check the following information for the resource. If it's not correct, please modify it.

Title:	The Semantic Header and Indexing and Searching		
Alt title:			
Author/Creator/Aggregator:	Bipin C Desai		(see comment to separate)
Publisher:			
Keywords:	Bibliographic record, Searching, Database system, In (see comment to separate)		
Version:			
Source:			
Language:	English		
Subject:	Click to add One or More Subject		
Identifier:	URL	http://www.cs.concordia.ca/~facul	
Classification:	Legal		
Coverage:	Audience		
System Requirement:	Software		
Created Date:	30	July	1997
Entry Date:	1	January	2004
Abstract:	<p>This paper describes an indexing system called semantic header for Internet resources. The semantic header contains the meta-information for each "publicly" accessible resource on the Internet. It also describes the registering system and the distributed database representing the union catalog of resources on the Internet. This database would be used in a search system to facilitate search.</p>		

Done Internet

Figure 5.2.1: Automatic resource registration interface (Part 1)

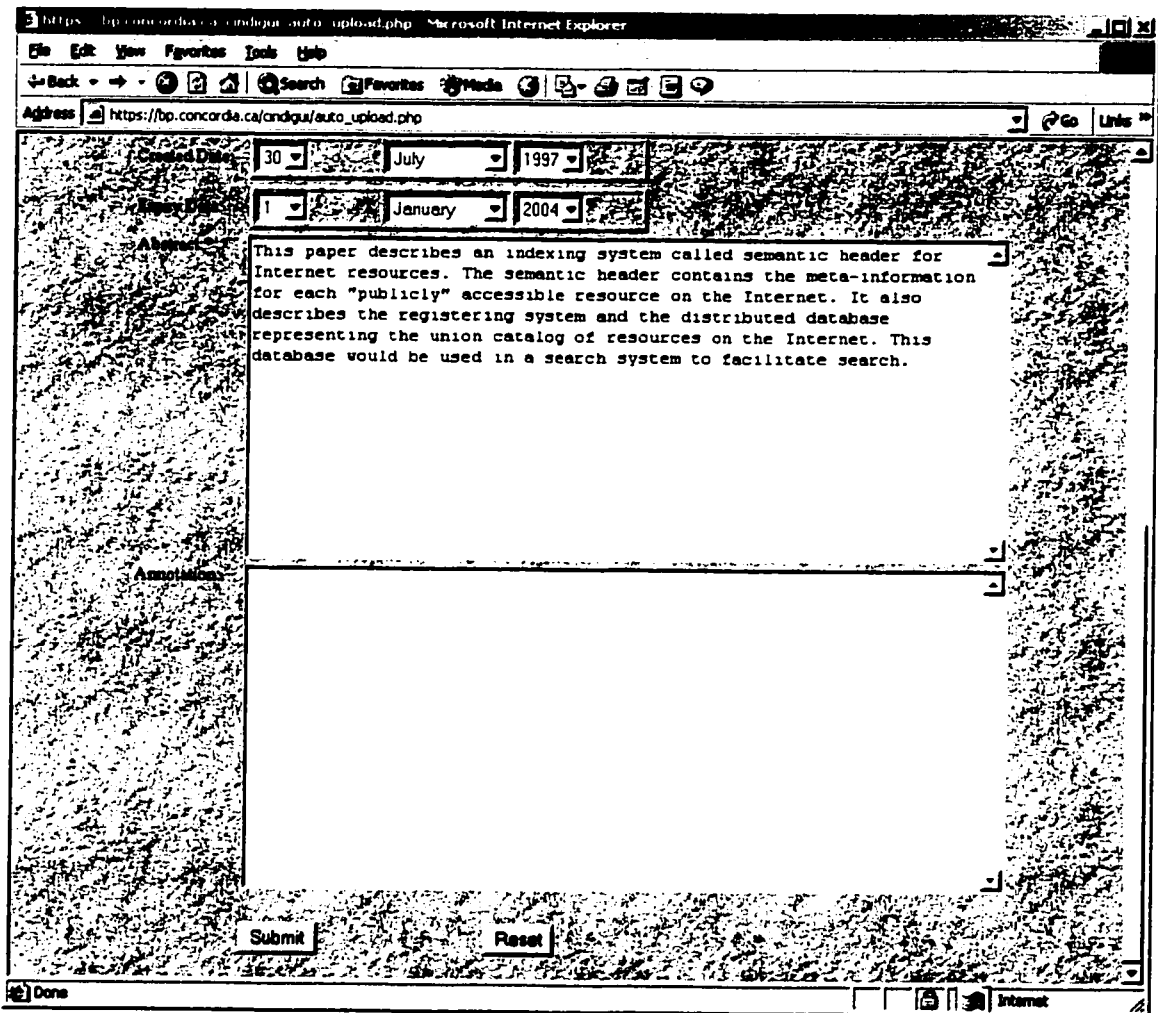


Figure 5.2.1: Automatic resource registration interface (Part 2)

Now one critical problem has to be solved: in the multi-user, multi-threaded web environment, how to match the uploaded file with its corresponding semantic header stored in the intermediate table, and display it again in step 5? Here we cannot just depend on the file name because two contributors may upload two different files with the same name simultaneously. Instead, we must invoke one unique identifier such as `resource_id`, combine it with the file name. Similar to the manual resource registration, the database system can automatically generate a unique `resource_id` with

AUTO_INCREMENT attribute for each new uploaded file. The multi-user safe function mysql_insert_id() can be used to retrieve the last resource_id. The following PHP/MySQL code shows the process how to generate the unique resource_id on a per-connection basis, how to find the corresponding semantic header from the intermediate table, and how to insert data into the SHDB.

```
// generate the unique resource_id
$result = mysql_query("insert into resource values
    (null,'title', null, 3, null, null, '0000-01-01','0000-01-01',
    '0000-01-01','0000-01-01', null, null, 10, null, 'abstract',
    'filename', 0, 0)");

if($result)
{
    $paperid = mysql_insert_id($db);

    $f_name=($paperid+1000000) . "_" . $userfile_name;
    $destination="/home/cindigui/www/papers/$f_name";
    $dest="$f_name";
}

if (copy($userfile,$destination))
{
    exec("/home/zzhang/www/register $dest");
    $filename = $dest;
}
.
.
.
// select the matched semantic header from the intermediate table
$result = mysql_query("select * from sh
    where filename='$filename'", $db);
.
.
.
// insert into the resource table
$result11 = mysql_query("UPDATE resource SET title='$title',
    alt_title='$alt_title',language_id=$language,
    keyword='$keyword', publisher='$publisher',
    created_date='$publish_date',upload_date='$date',
    expiry_date='$expiry_date', last_update='$date',
    version='$version', source='$source',size=$size,
    resource_format='$type', abstract='$abstract',
    filename='$filename' WHERE resource_id=$paperid" );

// insert record into table annotation
if ($annotation != "" )
    $result0 = mysql_query("insert into annotation values
        ($paperid,'xm','$annotation')");
```

```

// insert record into table identifier
if ($identifier != "" )
    $result1 = mysql_query("insert into identifier values
        ($paperid, '$identifier_type', '$identifier')");

// insert record into table coverage
if ($coverage != "" )
    $result2 = mysql_query("insert into coverage values
        ($paperid, '$coverage_type', '$coverage')");

// insert record into table system_req
if ($sysrequirement != "" )
    $result3 = mysql_query("insert into system_req values
        ($paperid, '$sysrequirement_type', '$sysrequirement')");

// insert record into table classification
if ($classification != "" )
    $result4 = mysql_query("insert into classification values
        ($paperid, '$classification_type', '$classification')");

// insert record into table resource_subject
if ($subject1 != 0 )
    $result5 = mysql_query("insert into resource_subject values
        ($paperid, $subject1, $sub_subject1, $sub_sub_subject1)");

```

The automatic resource registration interface in Figure 5.2.1 is very similar to the Figure 5.1, except an unlimited number of three-level subjects can be entered. After you press the “Add one or more subject” button, a new window will pop up and display a set of subjects generated by ASHG (see Figure 5.2.2). Now you can review all the three-level subjects and choose “Accept” or “Reject” for each of them, these accepted subjects will be stored into the SHDB. Figure 5.2.3 allows you to add more new three-level subject. The three-level hierarchy subject pull-down menu provides a standard bibliography catalog and a correct hierarchy of subject, sub-subject, and sub-sub-subject, which can avoid incorrect inputs. We tested one contributor registering several documents at the same time and found that the system can handle them concurrently.

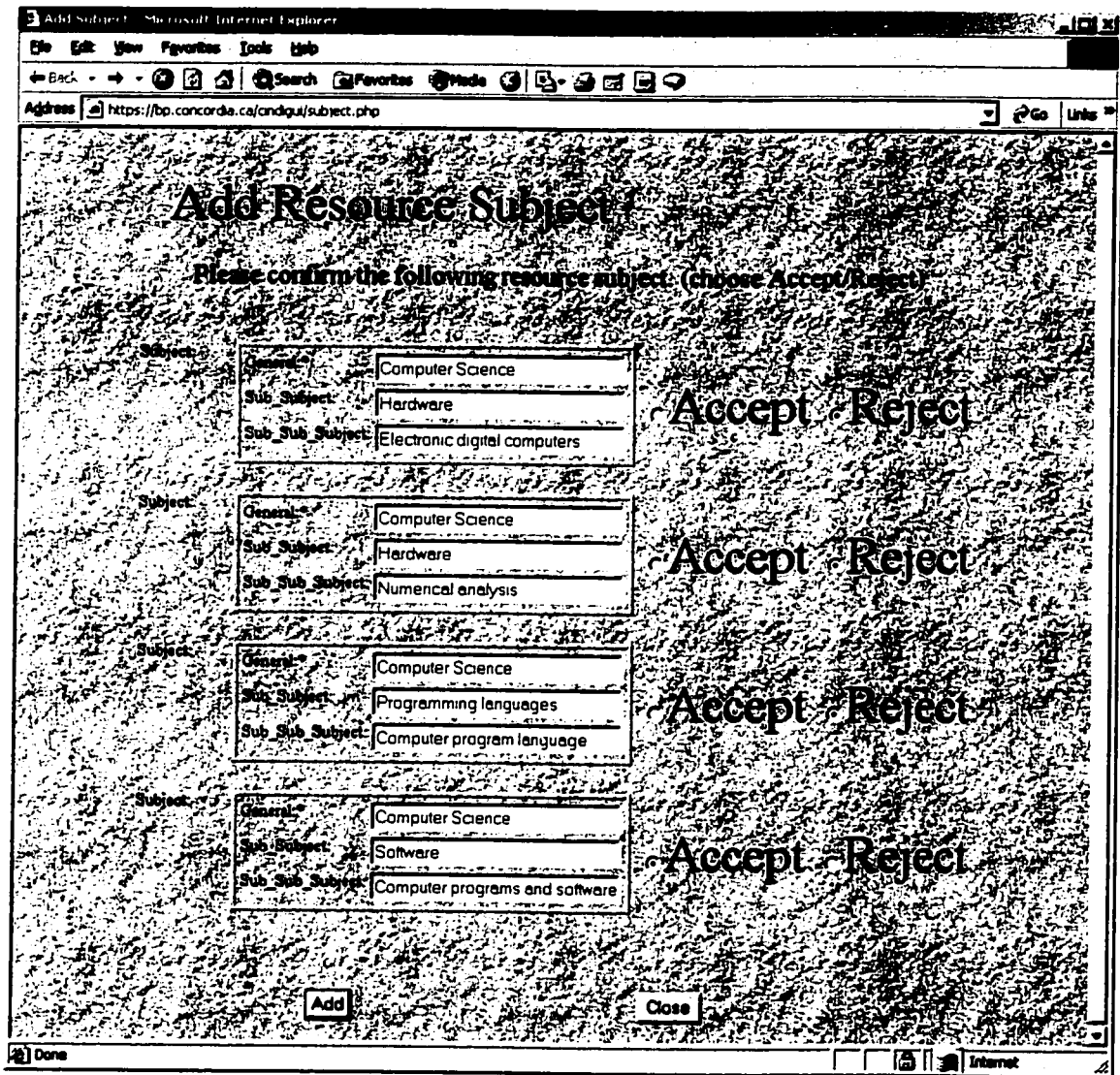


Figure 5.2.2: Automatic subject registration interface

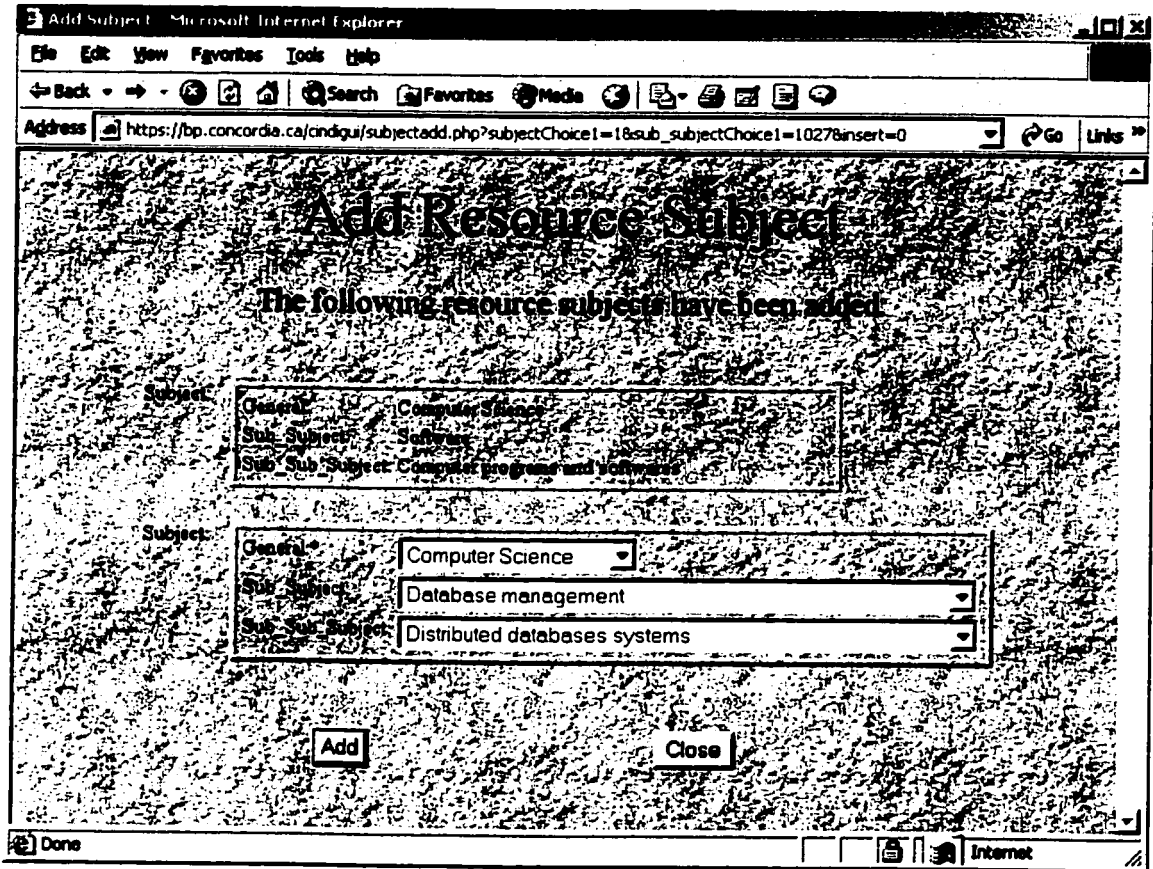


Figure 5.2.3: Automatic subject registration interface

5.3 Author Registration

After the contributor submits the resource registration form, the semantic header meta-information will be inserted into the Semantic Header database, and the message “The resource has been registered successfully!” will be displayed. Based on the input of the authors’ name field in resource registration interface Figure 5.1 or Figure 5.2.1, the system will ask the contributor to provide the corresponding personal information (see Figure 5.3). The author interface shows that the role field is required and all other fields such as organization, address, phone, Email, are optional. The system first checks the database to identify whether an author exists or not, if all the corresponding fields are

same, then no new record is inserted into the author table, instead, the author ID and the resource ID are inserted into the resource_author table. Otherwise, a new author ID is created and his personal information is stored in the author table, moreover, the new author ID and the resource ID are also inserted into the resource_author table. If there are many authors, the above procedure will be repeated until all of them are registered.

The author registration interface provides the opportunity to verify the authors' information for automatic resource registration. After the contributor successfully registers all the authors' fields one by one, the PHP program will display the message "All the authors' detail of this resource have been registered successfully!". We also provide a pull-down menu to help the contributor input the role field.

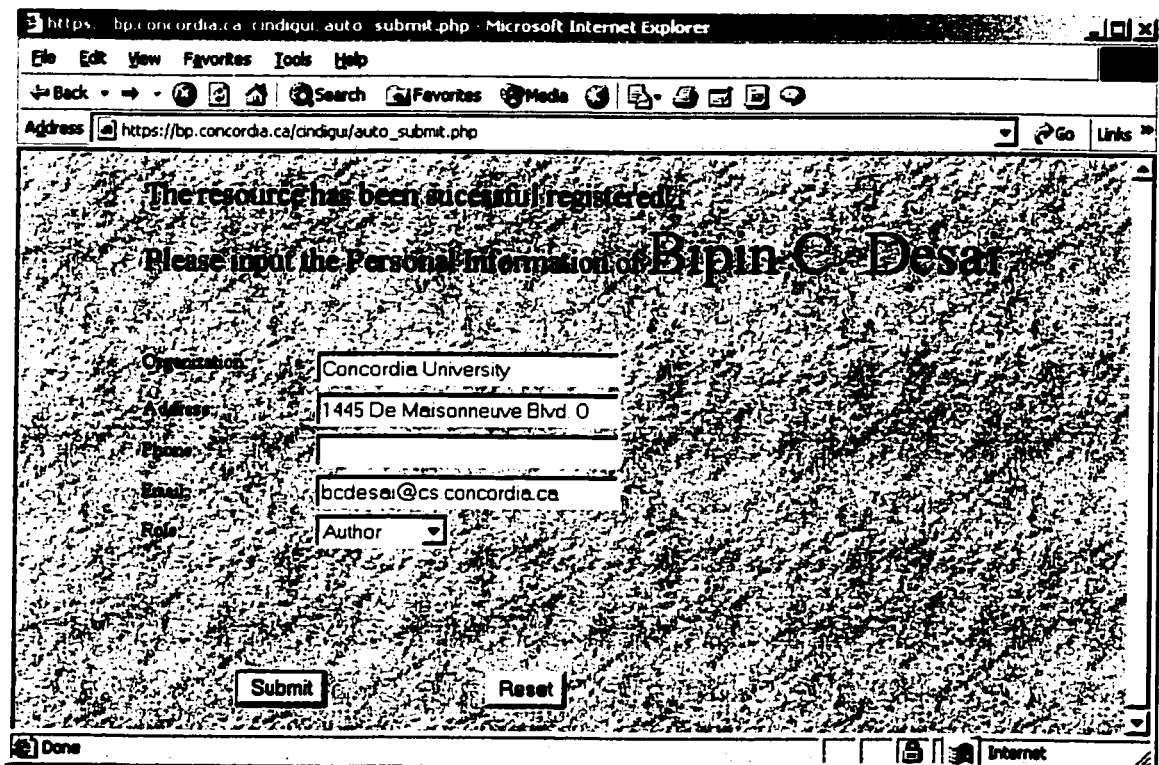


Figure 5.3: Resource author registration interface

Chapter 6

Search and Annotation Sub-systems

Resource Search Sub-system facilitates the user to find the useful information from the Semantic Header database. After the registered user successfully logs into the system, three different search levels (simple search, intermediate search and advanced search) are provided in Figure 6. The search query criteria are comprised of the title, author(s), keyword(s), subject and period of created date. The graphical user interface

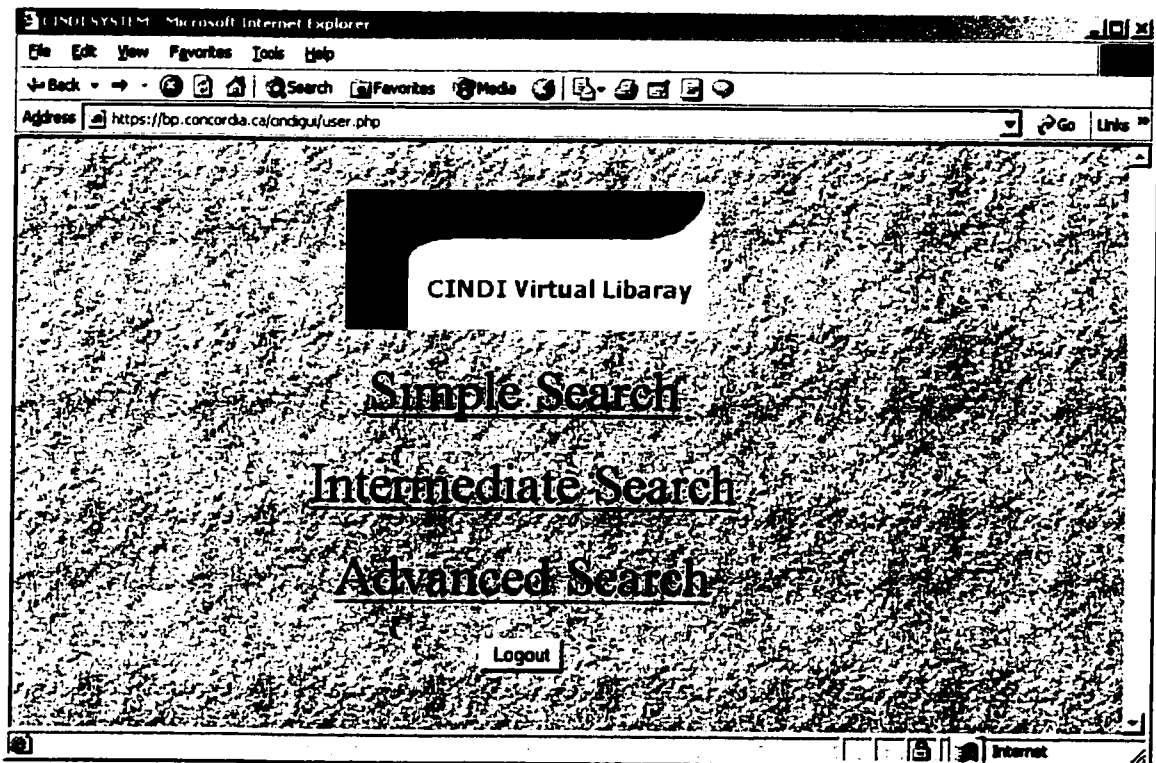


Figure 6: Search sub-system interface

(see the following section) allows the user to fill the query field, and pass it to the server as an SQL query string that will be performed on the backend database. Once the query is processed, the results will be returned to the client browser.

6.1 Search Query Structure

The following BNF grammar describes the search query structure which consists of a set of search criteria connected by the logical operator AND/OR. The logical operator precedence should be consistent with the Boolean expression of the SQL WHERE clause. In MySQL database, AND has higher priority than OR.

```

<search> ::= <operand> [ <op> <operand> [ <op> <operand>
           [ <op> <operand> [ <op> <operand> ] ] ] ]
<operand> ::= <title> | <subject> | <author_unit> |
             <keyword_unit> | <date>
<op> ::= <AND> | <OR>
<title> ::= <exacttitle> | <substringtitle>
<exacttitle> ::= <string>
<substringtitle> ::= <string>
<subject> ::= <general> | <general> <AND> <sub_subject> |
             <general> <AND> <sub_subject>
             <AND> <sub_sub_subject>
<general> ::= <string>
<sub_subject> ::= <string>
<sub_sub_subject> ::= <string>
<author_unit> ::= <author> | <AND | OR> <author_unit>
<author> ::= <string>
<keyword_unit> ::= <keyword> | <AND | OR> <keyword_unit>
<keyword> ::= <string>
<date> ::= <from_date> | <to_date> |
          <from_date> AND <to_date>
<from_date> ::= <day> - <month> - <year>
<to_date> ::= <day> - <month> - <year>
<string> ::= <character> | <character> <string>
<character> ::= a|A|b|B|c|C| ... |x|X|y|Y|z|Z|0|1|2| ... |7|8|9

```

```

<day>           ::= 1|2|3|4| ... |28|29|30|31
<month>         ::= 01|02|03| ... |10|11|12
<year>          ::= 1990|1991|1992| ... |2008|2009|2010

```

6.2 Simple Search

Figure 6.2 shows the user interface of the simple resource search. The user can enter search criteria, such as title, author name, keyword, subject and period of created date. For a given title, it can be an exact or a substring title. To be simple, only one title or one author name or one keyword is allowed. All these search criteria fields are connected by AND operator, the default field is null. For example, Figure 6.2 indicates that the query is based on the “Minimizing” as substring title, “Naveen Garg” as author name and “approximation” as keyword, the created period from “Jan. 1, 1998” to “Jan. 1, 2000”.

The corresponding SQL query clause is the following:

```

SELECT DISTINCT e.resource_id
FROM author d, resource_author c, resource_subject a, resource e
WHERE a.resource_id=e.resource_id and c.author_id=d.author_id
      and c.resource_id=e.resource_id
      AND e.title like "%Minimizing%"
      AND d.name like "Naveen Garg"
      AND(e.abstract like "approximation" OR e.keyword like
          "approximation")
      AND e.created_date>"01-01-1998" AND e.created-date<"01-01-2000";

```

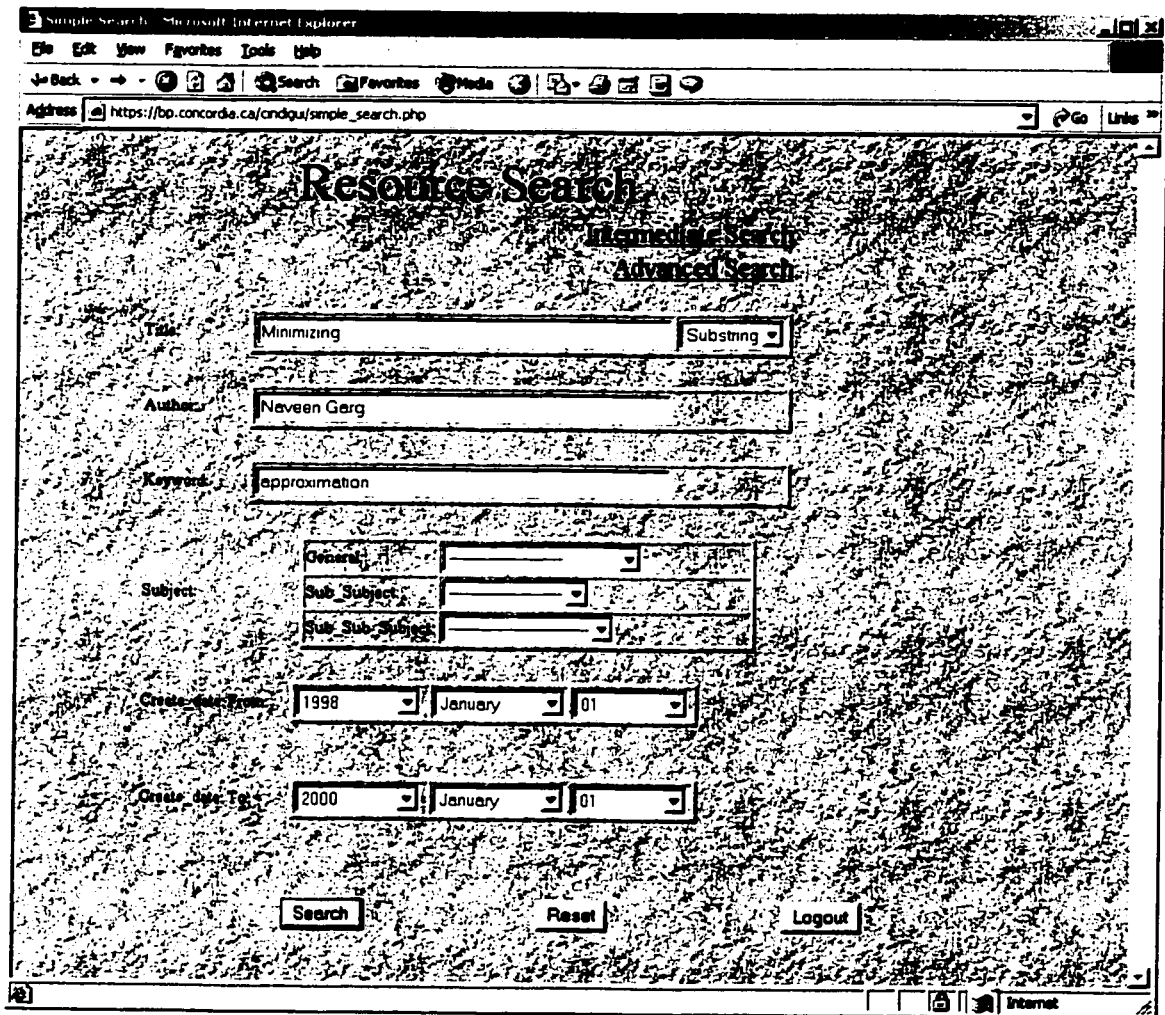


Figure 6.2: Simple resource search interface

6.3 Intermediate Search

Intermediate search interface (see Figure 6.3) allows the user to perform the more complex query. The given title can be an exact or a substring title. More than one author name can be entered, which are separated by comma and combined by AND or OR, but the AND/OR operator cannot be mixed. Similarly, many keywords can be separated by comma and combined by AND or OR, which are treated as substring for query. The user also can choose a subject at a general level, and/or corresponding sub_subject level

and/or corresponding sub_sub_subject level to perform a search. To be more flexible, the user can choose the AND or OR operator to connect these search criteria. For example, Figure 6.3 shows that the substring title is “Minimizing“, the author name is “Naveen Garg, John Hessen“, the keyword is “approximation, caching“, the corresponding SQL query should be:

```
SELECT DISTINCT e.resource_id
FROM resource_subject a, resource_authurname d, resource e
WHERE a.resource_id=e.resource_id and d.resource_id=e.resource_id
      AND e.title like "%Minimizing%"
      OR d.name like "Naveen Garg" OR d.name like "John Hessen"
      AND (e.abstract like "approximation" OR e.keyword like
           "approximation")
      AND (e.abstract like "caching" OR e.keyword like "caching")
      AND e.created_date>"01-01-1998" AND e.created_date<"01-01-2000";
```

Since Current MySQL version doesn't support view and set operations such as INTERSECT, UNION, we have to introduce redundant resource_authurname table to perform the multi-author search criteria. On the other hand, the redundant table will speed up the search performance.

```
create table resource_authurname (
      resource_id      int(10) unsigned NOT NULL,
      name              varchar(255)  NOT NULL,
      INDEX i_resource_authurname_rid (resource_id),
      primary key(resource_id),
      FOREIGN KEY (resource_id) REFERENCES resource(resource_id))
TYPE=INNODB;
```

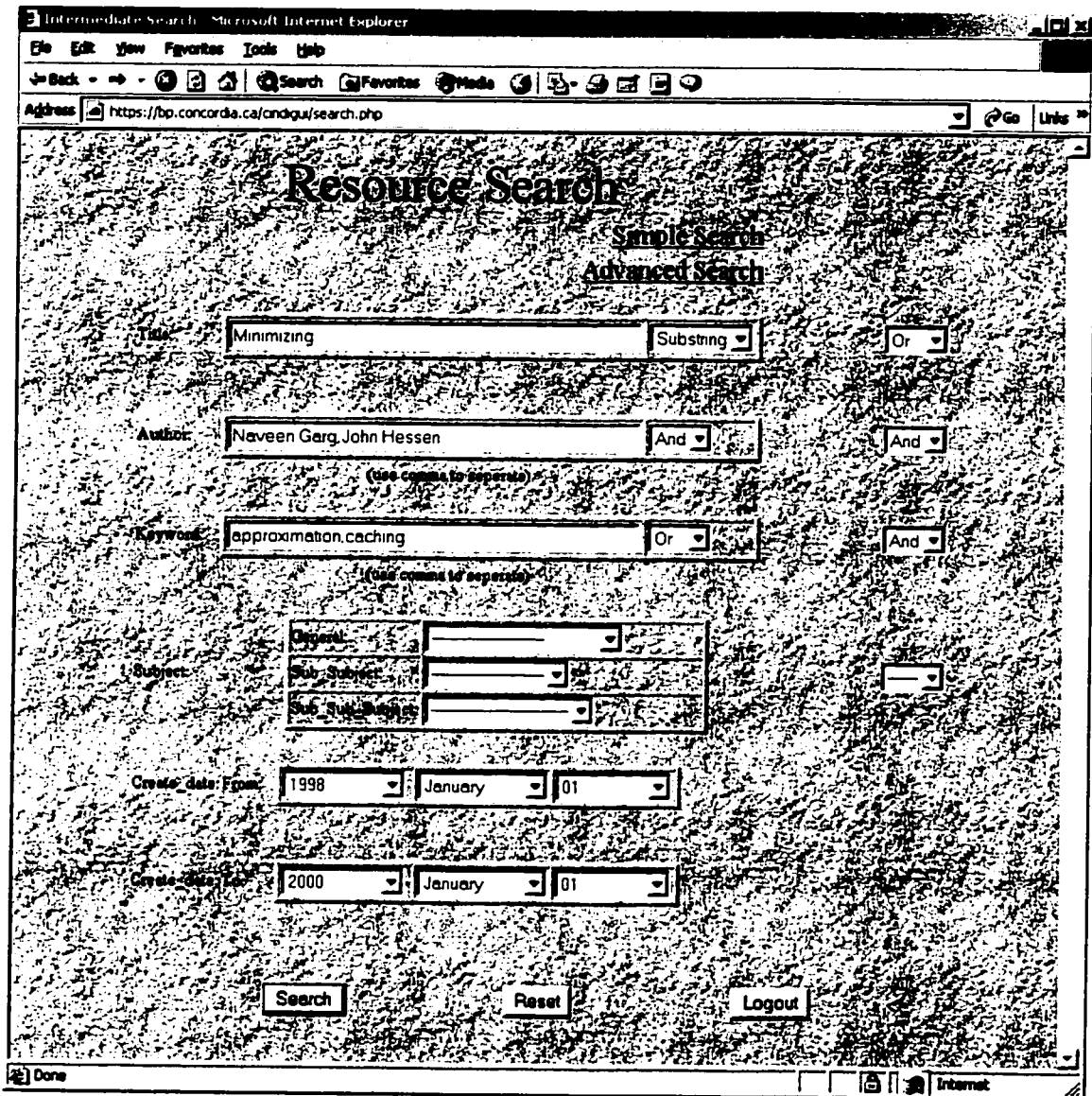


Figure 6.3: Intermediate resource search interface

6.4 Advanced Search

Figure 6.4.1 shows the advanced resource search interface, which allows the user to construct the author names or keywords by using the mixed AND/OR operators. For example, to enter a more complicated query as the following:

To search for Semantic header for resources written by both "Naveen Garg"

and “John Hessen” or written by “Larry Tom”.

The corresponding steps would be:

1. Enter “Naveen Garg” in the Author field.
2. Click the “Add One or More Author” Button.
3. Select AND operator.
4. Enter “John Hessen” in the Author field.
5. Click the “Add One or More Author” Button.
6. Select OR operator.
7. Enter “Larry Tom” in the Author field.

Similarly, we also can construct more complicated keywords combination by following the same procedure.

After the user enters the query and presses the “Search” button, the system will display the search results by page. Each page contains three resource semantic headers, which include the title, author(s), three-level subject(s), keywords, abstract, document, hit number and annotation (see Figure 6.4.2). The user can click the page number to enter the right place. Moreover, the system provides the corresponding link in the document field for users to either download the resources or browse the resources in detail. If no matched resource is found, the system also will inform the user.

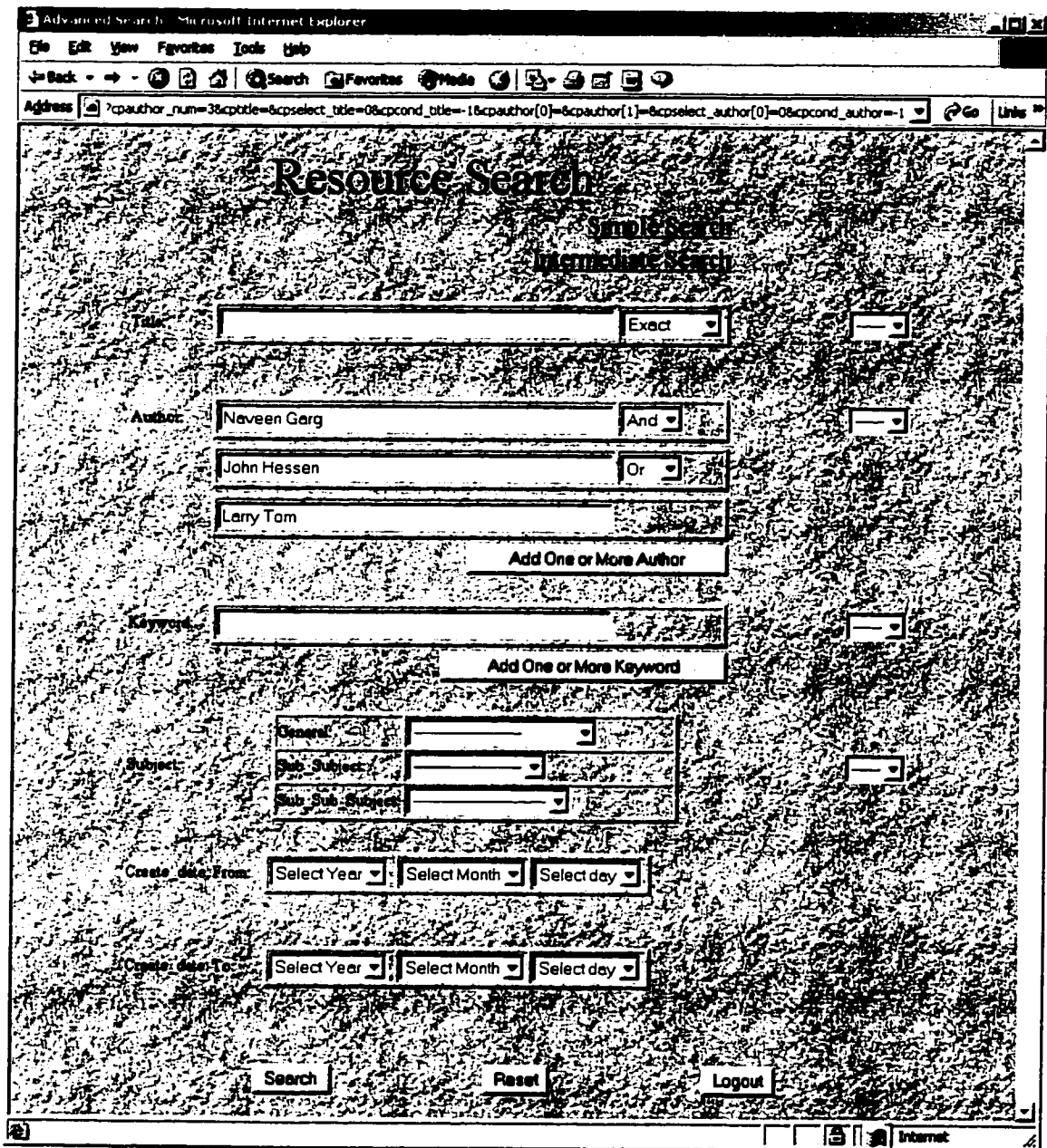


Figure 6.4.1: Advanced resource search interface

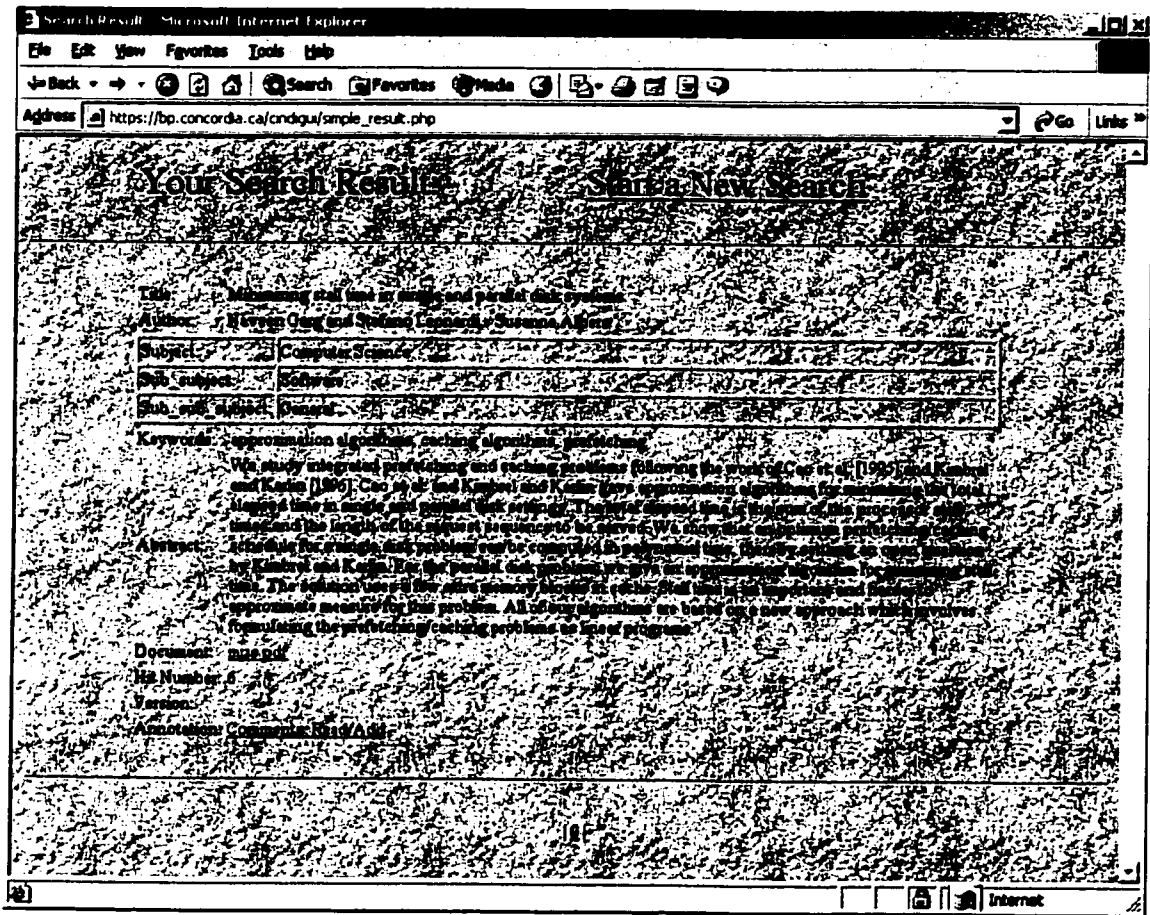


Figure 6.1.2: Resource search results

6.5 Annotation Sub-system

As shown in Figure 6.4.2, there is an “annotation” link. When the user clicks the link, an annotation window will pop out (see Figure 6.5). The annotation subsystem interface allows the user to view the existing annotations or to make comments on this resource. After the current user enters the annotation and presses the “submit” button, the system will store the annotation into the annotation table with his user ID and the resource ID.

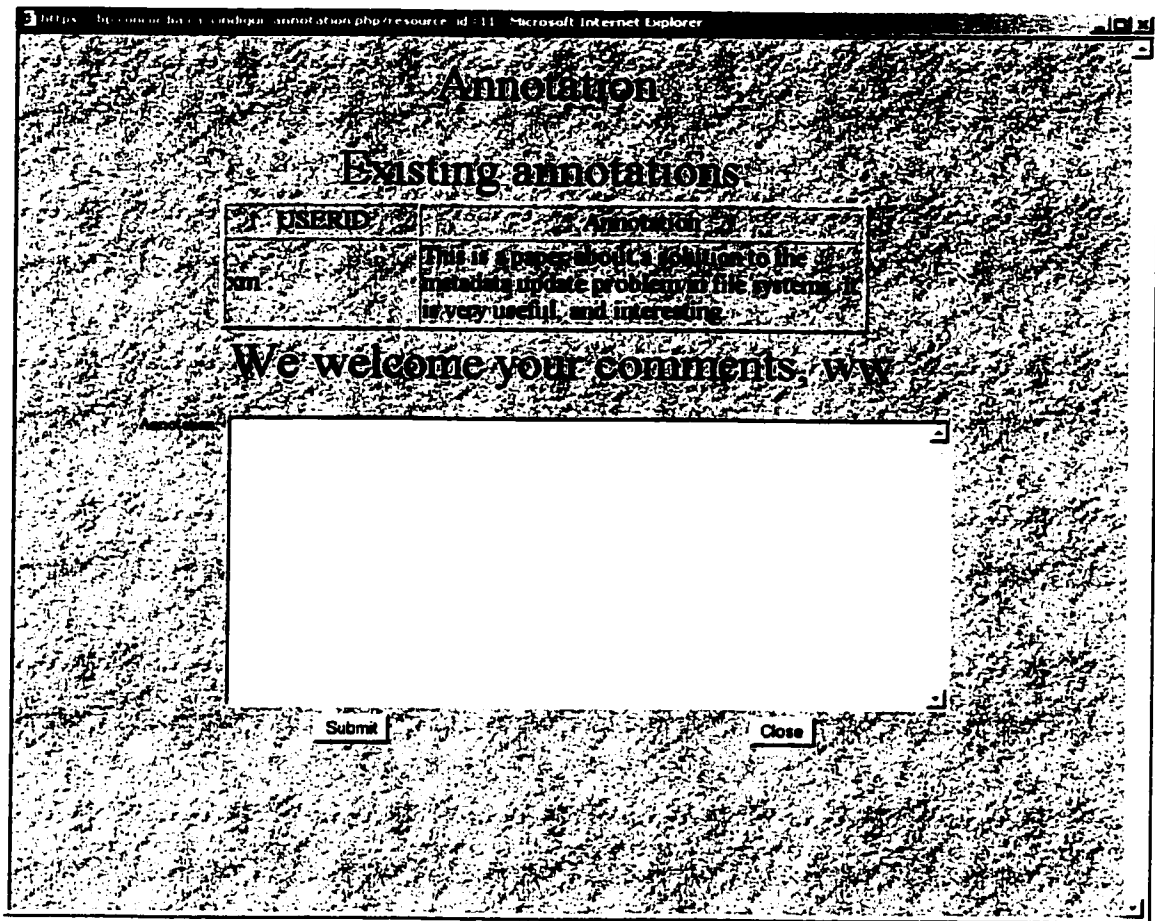


Figure 6.2: Annotation subsystem interface

6.6 Error Handling for Search Sub-system

The system formulates the search query based on the current value of the fields entered. To make sure that the final SQL query is correct, additional checking for each field must be performed. For example, the system must guarantee that at least one of these fields has been entered. If none of the search criteria is given, error message (see Figure 6.6.1) will warn the user. For intermediate and advanced search, we also provide more intelligent check for the logical errors. If both the title and author field are empty, the user only filled the keyword field and choose AND/OR operator between these fields,

error message, as shown in Figure 6.6.2, will inform him that the title or author criteria are needed.

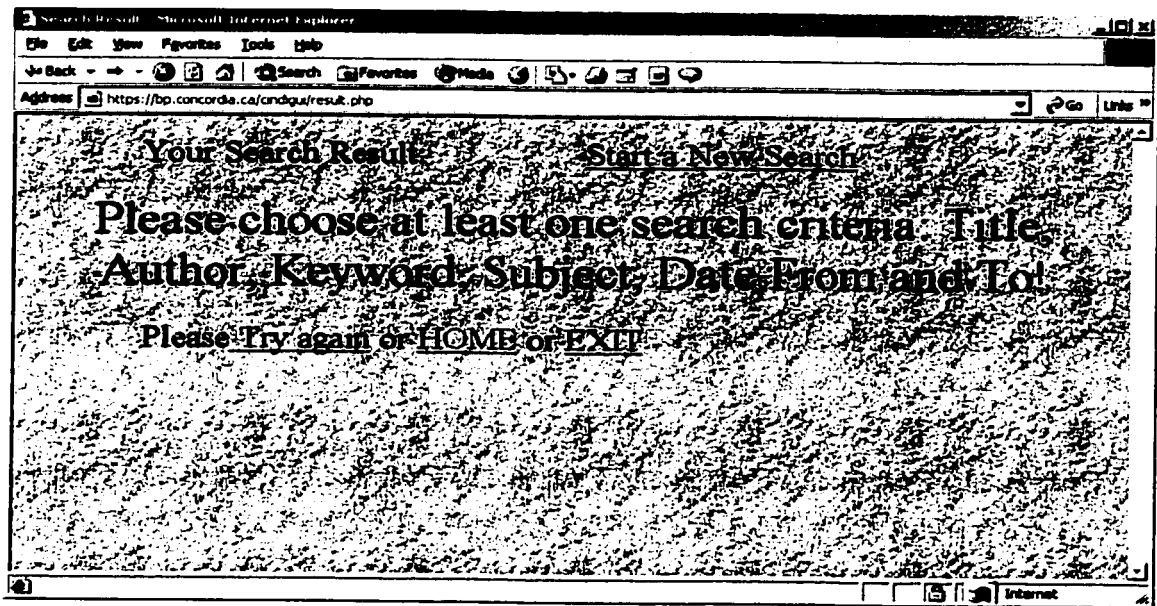


Figure 6.3.1: No search criteria error

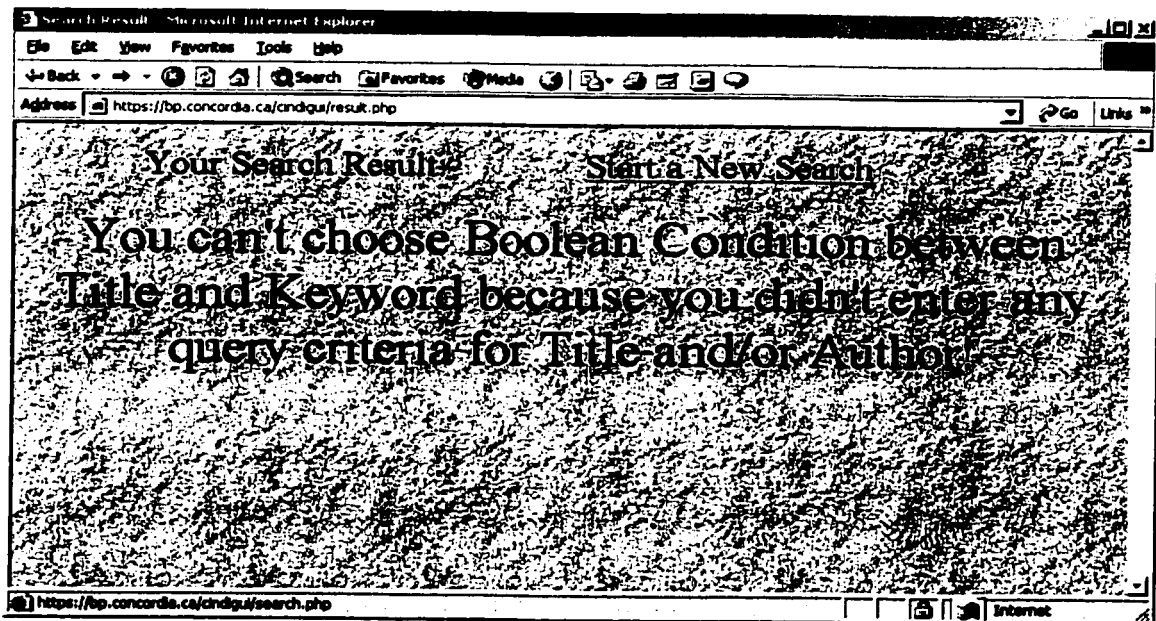


Figure 6.6.2: Query Boolean condition error

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The web-based CINDI system is developed to enable the user to find the useful hypermedia documents through the Internet. It tries to build an efficient and effective virtual library by using the standard index system, i.e. Semantic Header, a data structure to record the meta-information of network resources.

The Semantic Header meta-data entry can be provided either by the resource contributor or by invoking the Automatic Semantic Header Generator (ASHG) program. The former method would be more accurate whereas the latter is much faster and can cover a large amount of resources. The presence of the abstract in the Semantic Header provides a summary of the document, which can help the user make better decision regarding the relevance of the resource. In order to facilitate the user to enter the resource Semantic Header information, the graphic resource registration interface has been designed and implemented, which contains the three-level hierarchy subject and other domains pull-down menu.

Most of the existing search engines (such as Google, AltaVista) retrieve the information only by keywords. Since the same keyword may have different meaning in different domains and places, this kind of scheme may cause lots of miss-hits. The CINDI system solves this problem by providing the user several different search criteria: title, author name(s), keyword(s), subject and period of created date. The graphical user interface for resource search subsystem allows the user to find the needed documents efficiently from the CINDI virtual library.

7.2 Contribution of this Report

The contributions made by this major report to the CINDI system are listed as follows:

- Introduce InnoDB table to MySQL database to support the transaction and the foreign key constraint, then redesign and implement the Semantic Headers database sub-system.
- Design and implement the resource search sub-system, which includes the three level search strategy, the results display interface and more intelligent error handling mechanism. By optimizing the query, a better search performance is achieved.
- Redesign and implement of resource registration sub-system: manual or automatic registration. Support multi-user to register the resource semantic header concurrently by invoking InnoDB transaction model.
- Integrate the Automatic Semantic Header Generator sub-system with the resource registration sub-system.

- Allow the user to input unlimited three-level subjects for automatic resource registration, design and implement the subject entry registration interface.
- Improve the security control sub-system with PHP session.

7.3 Future Work

In order to greatly improve our system's performance and efficiently retrieve useful information from the Internet, the following section will provide some suggestions for future work based on the current web-based CINDI system.

1. Building the distributed database system

A distributed database is a set of databases stores on different sites connected with network, but typically appears to applications as a single database. An application can transparently access or modify the data in the distributed database. Database replication will copy and maintain database objects in multiple sites that make up a distributed database system. Most commonly, replication is useful to improve the performance and protect the availability of applications because alternate data access option exists; so, if one site failed, we still can retrieve the same data from another site that contains a replica.

In this project, we have built a centralized database. One of the major future tasks is to design and implement a Semantic Header Distributed Database System. The database can be horizontally partitioned on subject areas. For example, we can store the semantic headers related to Computer Science in one site, and Electrical Engineering related semantic headers in another site. When the end users register a resource or perform a search query, the distributed database system will take the

responsible for deciding where to store the semantic header or where to find the required information. This distributed and replicated nature of the Semantic Header database can provide reliability and scalability.

Oracle offers the implementation of a distributed database. Each oracle database in a distributed database system is controlled by its local server, which cooperates to maintain the consistency of the global distributed database. Unfortunately, current MySQL version does not support this distributed feature.

2. Tune the database structure

Current MySQL version doesn't support view and set operations such as INTERSECT, UNION, so we have to invoke the redundant resource_authname table to perform the multi-author search. If the new version can support the set operation, you can delete the resource_authname table. But the redundant table will help speed up the search performance.

3. Add auxiliary functions

Both the contributor and user can change their passwords and other personal information. In case one person forgets his password, the system can send him the password via the email provided during registration by checking the login ID. From the point of view of the database administrator, we should provide a set of graphical interfaces to maintain the Semantic Header database.

4. Add Online help

Online help provides an easy way for users to learn how to efficiently use this system. It should include the detailed description for each sub-system, the user interfaces and what kind of input is expected for the required fields. Its importance cannot be ignored for our CINDI system.

References

- [1] Bipin C. Desai, The Semantic Header and Indexing and Searching on the Internet, <http://www.cs.concordia.ca/~faculty/bcdesai/cindi-system-1.0.html>

- [2] Bipin C. Desai, Test: Internet Indexing Systems vs List of Known URLs: Revisited, <http://www.cs.concordia.ca/~faculty/bcdesai/web-publ/test-of-index-systems-revisited.html>

- [3] Bipin C. Desai, Shinghal Rajjan, A System for Seamless Search of Distributed Information Sources, May 1994. <http://www.cs.concordia.ca/~faculty/bcdesai/web-publ/w3-paper.html>

- [4] Bipin C. Desai, Report of the Metadata Workshop, Dublin. March 1995. <http://www.cs.concordia.ca/~faculty/bcdesai/metadata/metadata-workshop-report.html>

- [5] MySQL Documents, <http://www.mysql.com/documentation/>

- [6] MySQL Bechmarks, <http://www.mysql.com/information/benchmarks.html>

- [7] Client/Server Software Architecture, http://www.sei/cmu.edu/str/description/clientserver_body.html

- [8] PHP Documents, <http://www.php.net/docs.php>

- [9] Bipin C. Desai, *An Introduction to Database Systems*, West St Paul, 1990.

- [10] Raghu Ramakrishnan, *Database Management Systems*, McGraw Hill, 1998.
- [11] Apache secured by SSL, <http://www.apache-ssl.org>
- [12] Larry L. Peterson and Bruce S. Davie, *Computer Networks: a Systems Approach*, Morgan Kaufmann, 2000.
- [13] Wen Tian, Web Based CINDI System: Database Design and Implementation, Department of Computer Science, Concordia University, 2001.
- [14] Xiaomei Yang, Web Based CINDI System: Graphical User Interface Design and Implementation, Department of Computer Science, Concordia University, 2001.
- [15] Mohamed Amokrane Mechouet, Web Based CINDI System, Department of Computer Science, Concordia University, 2002.
- [16] Youquan Zhou, CINDI: The Virtual Library Graphical User Interface, Department of Computer Science, Concordia University, 1997.
- [17] Sami Samir Haddad, Automatic Semantic Header Generator, Department of Computer Science, Concordia University, 1998.
- [18] Leon Atkinson, *Core PHP programming: using PHP to build dynamic Web sites*, Prentice Hall PTR, 2001.