P-1601

INTELLIGENT QUERY ANSWERING (IQA) SYSTEM

A PROJECT REPORT

Submitted By

AYSHWARIA.K.B **(71202104004)**

KRITIKA.R **(71202104018)**

In partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2006

BONAFIDE CERTIFICATE

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"INTELLIGENT QUERY ANSWERING (IQA) SYSTEM"** is the bonafide work of **"Ayshwaria.K.B (71202104004)** and **Kritika.R (71202104018)"** who carried out the project under my supervision.

SIGNATURE

Dr. S. Thangasamy

HEAD OF THE DEPARTMENT

Department of Computer Science & Engg.
Kumaraguru College Of Technology
Chinnavedampatti Post
Coimbatore – 641 006

SIGNATURE

Mrs. D. Chandrakala

SUPERVISOR

Assistant Professor
Department of Computer Science & Engg.
Kumaraguru College Of Technology
Chinnavedampatti Post
Coimbatore – 641 006

Submitted for viva-voce examination held on 02/05/2006

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

DECLARATION

We hereby declare that the project entitled **"INTELLIGENT QUERY ANSWERING (IQA) SYSTEM"** is a record of original work done by us and to the best of our knowledge; a similar work has not been submitted to Anna University or any institution, for fulfillment of the requirement of the course study.

The report is submitted in partial fulfillment for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place : Coimbatore

Date : 21.04.2006

K.B. Ayshwaria

(Ayshwaria.K.B)

Kritika R

(Kritika.R)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

With profound gratitude, we express our deepest thanks to our internal guide Mrs.D.Chandrakala, Assistant Professor, Department of Computer Science and Engineering, who has taken all measures to guide us through the project, and been a constant source of inspiration and motivation at various levels of the project.

Our sincere thanks to all the lab technicians who have been operational in aiding us implement the system.

We would like to thank the Head of the Department, Computer Science and Engineering, Dr. S. Thangasamy and Mrs. P. Devaki, Project Coordinator for guiding us through the project.

Our sincere thanks to the Department of Computer Science of Engineering, Kumaraguru College of Technology, for extending its fullest support by all means to enable us to complete the project.

Last but not the least, we extend our utmost gratitude to our parents, all our student peers, and all those who directly or indirectly helped us in successful completion of the project.

ABSTRACT

ABSTRACT

Any database query is used to find some answers from data stored in a database that meet some conditions or constraints of a retrieval statement. With the growing size of the database in the present days, it is required by the users to have the system to be intelligent in answering queries. Intelligent answers are those that do not provide wrong or misleading answers but in addition to providing the right answers also provide extra related information.

To meet this need of intelligent answering of English queries, we propose to develop an Intelligent Query Answering (IQA) System that enables easy retrieval of data from a database by means of a manually fabricated knowledge base. All types of queries can be answered by simple retrieval or intelligently by analyzing the intent of the query, neighborhood or associated information using the discovered knowledge base.

We have proposed a Query Rewriting Algorithm to transform the unstructured English query presented by the user, into the SQL query that can be executed directly on the database to retrieve the necessary details. We have also incorporated neighborhood concepts and intent analysis in order to provide more flexibility to the IQA System.

In order to demonstrate the above algorithm we have developed a system that uses the proposed algorithm to answer queries posed by the naïve users in order to retrieve details of a student. This Student database comprises of Personal Details, Academic Details, Placement Details, and Extracurricular Details and Club Membership Details of the different students.

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION

1. INTRODUCTION

A database query is to find so
database, which meet some conditions or constraints of a retrieval statement (e.g. Select statement). Database systems rely on an exact matching method for retrieving records (tuples) from a database. A record either satisfies the query or does not. A user is required to have detailed knowledge about the problem domain and some understanding about the database schema to properly construct a query. Moreover, the user is also expected to know the right format in which the query is to be presented.

Intelligent query answering refers to a procedure that answers queries constructed in simple English, effectively and intelligently by searching the query for specific keywords and values. An intelligent answer should be correct, non-misleading, and useful to a query. Besides, Intelligent querying also analyses the intent of a query and providing some generalized, neighborhood, or associated answers. Due to the complexity of the database schema, incorrect or incompletely specified queries are frequently posed and the users often receive no answers. With the increas
and complexity of database schemas, it is getting more difficult to understand the schema to issue proper query to get the expected answers from the databases.

Intelligent Query Answering (IQA) System that is designed consists of analyzing the intent of query, rewriting the query in SQL format and also providing additional data based on the intention and neighborhood, generalized or associated information, and providing intelligent answers to the query. It is now necessary to provide more powerful and sophisticated

query processing capability to meet the needs of users working with databases, with no proper knowledge about the schema of the database and format for construction of query.

1.1 SCOPE

The IQA system can be used to retrieve information from a large repository in order to answer the queries posed by the user. The system can handle queries framed using simple English language without the use of SQL formats. Apart from answering the queries, the system also helps the user to frame sensible queries by providing hints to construct queries. The IQA system answers to the queries by analyzing the intent of the user by means of providing links. The system cannot be used to update or delete entries in the repository.

The IQA system is developed in a much generic sense, by considering all the possible bottlenecks that would arise when the system is tried to be integrated with a different database, such that, the same implementation logic can be used for any database only by including relevant details in the mapping tables and making corresponding modifications in the code.

In order to demonstrate the working of the IQA model, we use a Student database. The Student database comprises of Personal Details,

LITERATURE REVIEW

2. LITERATURE REVIEW

2.1 EXISTING SYSTEM

In the existing query answering systems, the user is required to commit to memory all the commands and their syntax in order to obtain the required details from a database. Moreover, the answer for a posed query relies on an exact matching method for retrieving records (tuples) from a database. Besides, the existing system returns either too many or too few items or even no answer, as the query may be too general or too specific or does not match the databases schema. The user is required to have detailed knowledge about the problem domain and some understanding about the database schema to properly construct a query. The existing concept of query rewriting deals with optimized retrieval of result from database using views. But there are no systems that handle English queries and convert it into structured queries.

2.2 PROBLEMS IN EXISTING SYSTEM

- Retrieving tuples from multiple tables requires the user to have a detailed knowledge of the table relations in the database. (Natural-

- There is no means by which the user is informed about domain specific mistakes in query construction.
- Strictly, exact table names and field names have to be used in order to acquire the accurate matches for the constructed query.
- If the users are unable to remember the syntax even the help feature cannot be used effectively.

2.3 EXISTING METHODOLOGIES

Knowledge Discovery

The idea of intelligent query answering is achieved so far by using the concept of knowledge discovery and framing of knowledge rules as explained in [1] and [2]. These knowledge rules are stored in a repository and are used to answer queries in a more generalized manner. This is achieved by the usage of concept hierarchies and generalized rules. These rules are used to provide accurate as well as summary information relevant to the query presented. These queries presented should be constructed in a particular format to obtain the required answers.

Query Rewriting

The objective of answering queries uses the concept of query rewriting as expressed in [3]. Query Rewriting is used to form optimized queries from already existing views. This requires the existence of the table views in order to from the rewritten queries. The rewritten queries are executed over the views rather than the base relation to obtain the answers. This concept when used alone does not provide the facility of answering queries intelligently.

In the ***proposed methodology***, we have combined the concepts of knowledge discovery and query rewriting. Knowledge discovery is that which refers to manually fabricating a knowledge base of the database schema and domain related keywords. Query rewriting is the conversion of the English query to the SQL query. The main aspect of this concept is keyword analysis performed on the presented English query, so as to answer the query intelligently and interactively.

2.4 PROPOSED SYSTEM

The literature survey performed in the field of query answering revealed that, till date, there are no systems developed to handle English queries posed by the users in a user interactive way.

In order to handle this situation, we propose to develop an ***Intelligent Query Answering (IQA) System*** that provides the facility of user friendly querying from a database. We have proposed to achieve this objective by incorporating a manually created knowledge base of domain related keywords and the concept of Query Rewriting. To add flexibility to the system to handle a large variety of queries with no restriction of format, we use the concept of Neighborhood and Intent Analysis.

To implement the concept of Query Rewriting, we have put forward a new algorithm, the ***"Query Rewriting Algorithm"***. The Query Rewriting Algorithm is intended to convert an unstructured English query to a SQL query with minimum access to the database.

The ***Neighborhood*** concept is aimed at improving the flexibility of the system in answering a wide range of queries appropriate to the domain. This concept is incorporated in the algorithm by means of using a ***Neighborhood Table*** which has all the keywords and its synonyms pertinent to the domain considered.

Intent Analysis is targeted in achieving the objective of making the IQA system user-friendly. The concept of intent analysis comes up only when the query is too generalized or when more details relevant to the query are available to be provided to the user.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- The users of the IQA System need not have a complete understanding of the schema of the database.
- The users are not required to know the format or the structure in which the query has to be constructed in order to obtain the desired information from the database.
- The users are only required to know what information they want to retrieve or know from the database.
- The system also provides extra information relevant to the query apart from providing the details requested by the user.
- The system generates an intelligent answer that explains why the query fails thereby exposing any false presuppositions the user may have.
- The user preference such as intentions, interests and needs in databases are considered while answering the queries.

PROPOSED LINE OF ATTACK

3. PROPOSED LINE OF ATTACK

The implementation of the proposed system is split into various modules such as Neighborhood Module, Basic Identification Module, Combination Identification Module, Query Rewriting Module and Intent Analysis Module.

Each of these modules performs its required functions efficiently by using the various implementation-tables namely, DataDictionary, TableDetails, Neighborhood Table and Combinations Table. The first four modules are integrated by the Query Rewriting Algorithm.

The platform on which the IQA System is developed is Windows XP with the source coding done using ADO.NET and the front-end designed using ASP.NET. The database is designed and structured in SQL Server 2000.

PROPOSED METHODOLOGY

4. PROPOSED METHODOLOGY

A process model for developing any project is chosen based on its nature and application, methods and tools to be used, controls and deliverables that are required. The proposed line of attack for ***"Intelligent Query Answering (IQA) System"*** is ***Conversion of English Query to SQL Query*** and the objective is achieved using one of the significant software engineering models called the ***"Incremental Model".***

The Incremental Model combines the elements of Linear Sequential Model or Waterfall Model with the iterative philosophy of Prototyping. The Incremental Model delivers software in small but usable pieces called ***"Increments".*** In general, each increment builds on those that have already been delivered. Early increments are stripped down versions of the final product, but they do provide the capability that serves the user and also provide a platform for evaluation by the user.

When an incremental model is used the first increment is called the ***'CORE PRODUCT'***. Here the basic requirements are addressed, but many supplementary features remain undelivered. After evaluation a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and delivery of additional features and functionalities. The process is repeated following the delivery of each increment, until the complete product is produced.

4.1 INCREMENTAL STAGES IN THE DEVELOPMENT OF IQA SYSTEM

The ***core product*** of the IQA System consists of the database, domain-specific implementation-tables namely ***DataDictionary*** and ***TableDetails.*** The basic forms were designed to provide a suitable ***GUI.*** The ***Basic Identification Module*** uses the two implementation-tables to answer basic queries which have the exact keyword matches to table names and field names without value and conditional constraints can be effectively processed at this stage.

The ***second increment*** incorporates the addition of another implementation-table called ***Neighborhood Table.*** The ***Neighborhood Module*** which uses this table is included to enhance the flexibility of handling wide range of queries posed by the user. This increment was tested and found to be successful by verifying the systems ability to answer queries which has words that are synonymous to the exact keywords or field names.

The ***third increment*** is intended to handle queries with value and conditional constraints. This required an additional implementation-table, ***Combination Table.*** The ***Basic Identification Module*** is ***extended*** to use the values constraints specified in the query to identify the appropriate tables and fields. The ***Combination Module*** uses the Combination Table to identify the operators mentioned in the query in order to construct the suitable conditions. This increment was tested and found to be effective in answering queries with conditional and value constraints.

The ***fourth increment*** is aimed to consider user preferences and provide additional information the user. For this purpose, we included an additional form to display the various options relevant to the query and the corresponding processing is done in the ***Intent Analysis Module***. This increment was tested and found to display the relevant options. When any option is selected, the already posed query along with the selected option is processed and the exact tuples are displayed.

The Query Rewriting Algorithm is implemented in stages in each and every increment.

PROGRAMMING ENVIRONMENT

5. PROGRAMMING ENVIRONMENT

5.1 HARDWARE REQUIREMENTS

Processor	: AMD Sempron
Processor speed	: 137 MHz
RAM	: 256 MB
Hard Disk	: 10 GB
Mouse	: 3 button Scroll mouse
Monitor	:
Keyboard	: 101 keys enhanced

5.2 SOFTWARE REQUIREMENTS

Operating Systems	: Windows 9x/NT/XP/2000
Front End	: ASP.NET
Back End	: SQL Server 2000
Web Browser	: Internet Explorer 6.0

ARCHITECTURE OF IQA SYSTEM

6. ARCHITECTURE OF IQA SYSTEM

The IQA System consists of two important components namely,

Query Interface

This is the front end of the system which interacts with the user. This enables the user to pose his English query. The rewritten query and the corresponding result are displayed to the user by this Query Interface. This is also used to inform the user, about the mistakes in the posed query and also provide hints to construct meaningful query, thereby helping the user to obtain the required details.

Query Answering Engine

This is the important component of the IQA System. This is the one that does the English query processing. The processing involves two steps namely,

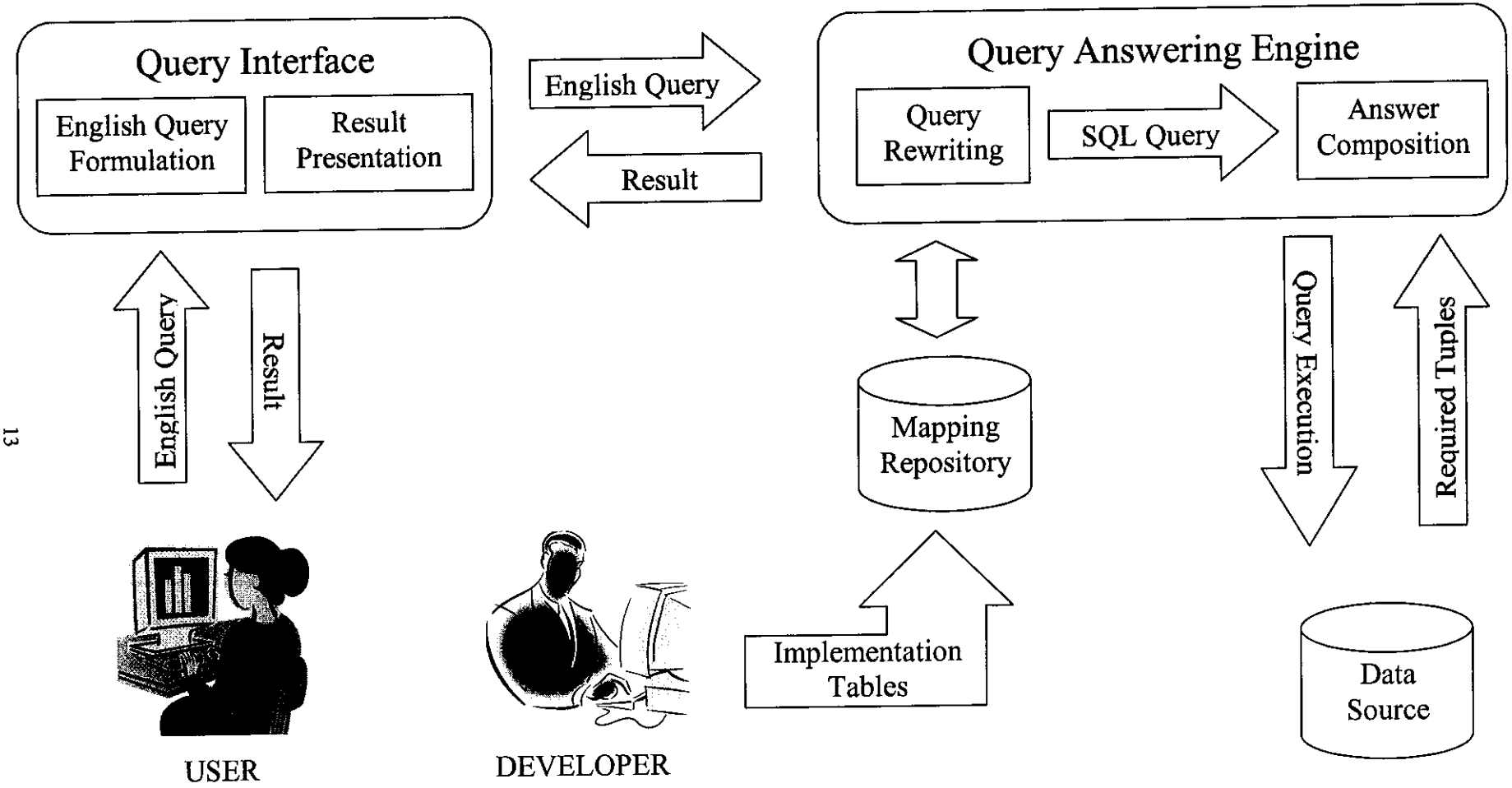
Query Rewriting

This step uses the proposed ng A accesses a set of implementation tables, relevant to the application and domain, that are consolidated to form the Mapping Repository as shown in Fig No. 6.1. These tables contain domain related keywords and database schema. With the help of these tables the English query is rewritten to form the structured SQL query.

Structured Query Execution

The constructed SQL query is executed on the database to obtain the required result. The relevant tuples are sent to the Query Interface and displayed to the user.

Fig No.6.1 Architecture of IQA System



IMPLEMENTATION DETAILS

7. IMPLEMENTATION DETAILS

7.1 QUERY REWRITING ALGORITHM

Input: Unstructured English query

Output: Structured Query

Processing:

Step 1: Split the query

Step 2: Replacing synonymous word with the right keyword or meaning.

Input: (i) A Query Q split into tokens w_i ($0 < i < l$), l is the number of word in the query (ii) An implementation table, the *Neighborhood table*

Output: The Query Q with the right keywords.

Method:

for each word w_i ($0<=i<=l$) of Q **do**

    for each row R in the *Neighborhood Table* **do**

        if $w_i = R_n$ ("*Synonym*") **then**

            Replace w_i with R_n ("*Word*");

Step 3: ***Identify the tables using keyword references***

Input: (i) A Query Q split into tokens w_i ($0 < i < l$), l is the number of word in the query, each token has specific keyword or values (ii) An implementation table, the *DataDictionary*

Output: A dataset *tabset* with the specific keyword and the table that the keyword refers to.

Method:

for each word w_i in the query **do**

for each row R_d in the *DataDictionary* table **do**

if $w_i = R_d$ ("*Keyword*") **then**

Include R_d in a dataset *tabdet*

Step 4: ***Identify the tables and fields using field references***

Input: (i) A Query Q split into tokens w_i ($0 < i < l$), l is the number of word in the query, each token has specific keyword or values (ii) An implementation table, the *TableDetails*

Output: A dataset *fielddet* with the specific fieldname and the table in which the field appears.

Method:

for each word w_i in the query **do**

for each row R_t in *TableDetails* table **do**

if $w_i = R_t$ ("FieldName") **then**

Include R_t in a dataset *fielddet*

Step 5: ***Identify tables and fields using value references***

Input: (i) A Query Q split into tokens w_i ($0 < i < l$), l is the number of word in the query, each token has specific keyword or values (ii)

Output: A dataset *valtab* with the value, specific fieldname and the table in which the value appears.

Method:

for each word w_i in the query **do**

for each table T in the database **do**

for each cell C in T **do**

if w_i = C **then**

Include the C and the corresponding Column Name and Table Name to a dataset *valtab*

Step 6: Using the above findings form the final list of tables and fields that will be needed in order to answer the query

The list of tables needed to answer the query is finalized by consolidating the tables in the datasets *tabdet* and *fielddet* in a dataset *finaltablst*. The final list of fields is obtained by consolidating the fields in the *fielddet* and *valtab* in a dataset *freshfielddet*. Care is taken to eliminate duplicate entries in the datasets.

Step 7: Form the SQL query by including the identified tables, fields and values in the appropriate clauses.

Input: (i) The identified datasets *fielddet, finaltablst, valtab*

Output: A dataset *valtab* with the value, specific fieldname and the table in which the value appears.

Method:

//Select Clause

For each row R_{fd} in *fielddet* **do**

Include R_{fd} ("FieldName") to the *Select Clause*

//From Clause

For each row R_{td} in *finaltablst* **do**

Include R_{td} ("TableName") to the *From Clause*

//Where Clause

For each row R_{vt} in *valtab* **do**

Form the condition R_{vt} ("FieldName") = R_{vt} ("Value")

Include the condition to the *Where Clause*

Step 8: If comparison operations are specified in the query, then identify the fields and values involved in the operation and append the relevant operator conditions to the query, in the where clause.

For each word w_i in the query **do**

If w_i is an operator *op* **then**

Identify the fields and values involved in the operation

Form the condition *FieldName op Value*

Include the condition in the *Where Clause*

Step 9: If additional information can be provided then specify the option to the user using links so that query answering can be done by analyzing the intent of the user

7.2 DATA FLOW DIAGRAM

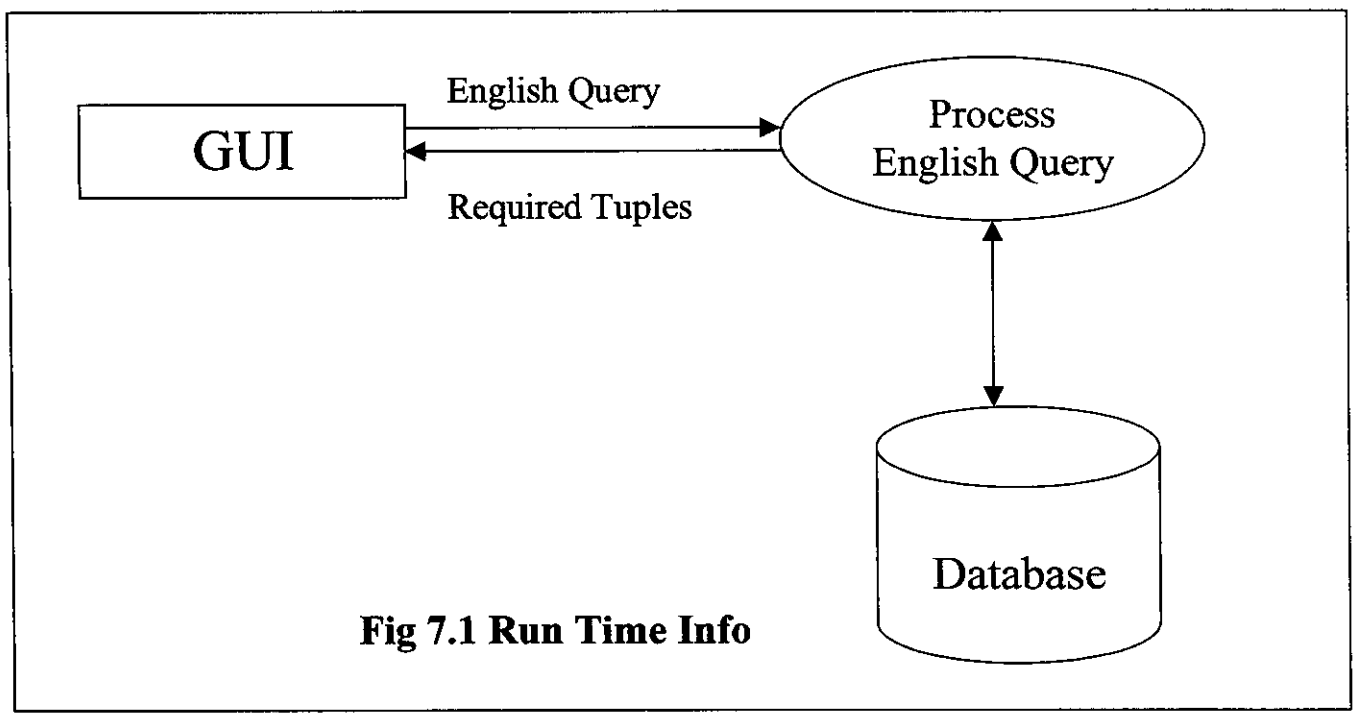


Fig 7.1 Run Time Info

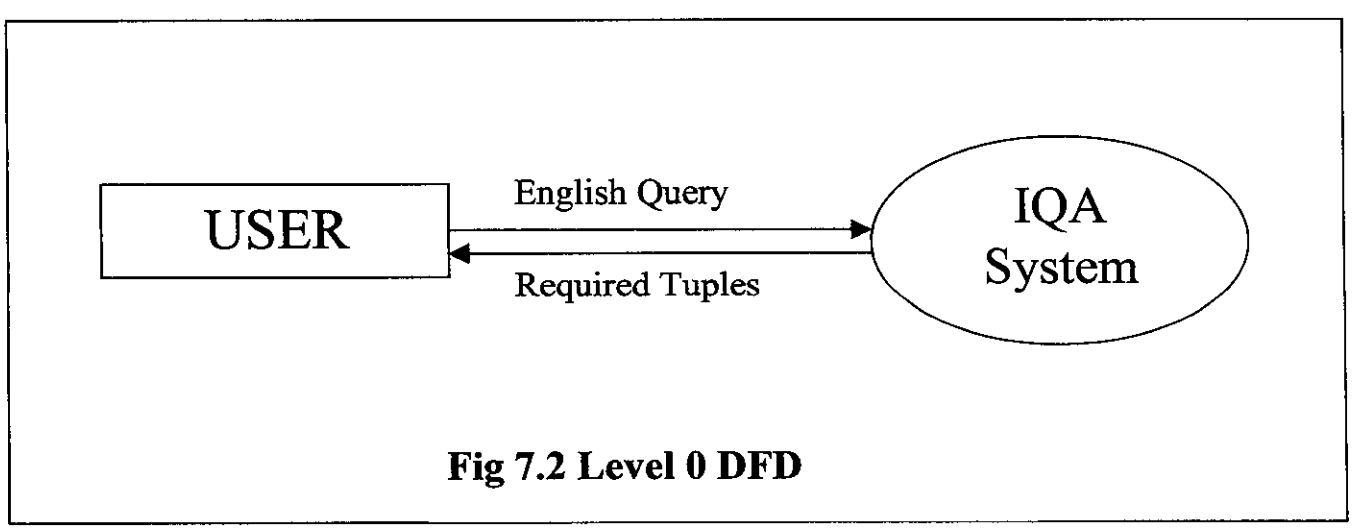


Fig 7.2 Level 0 DFD

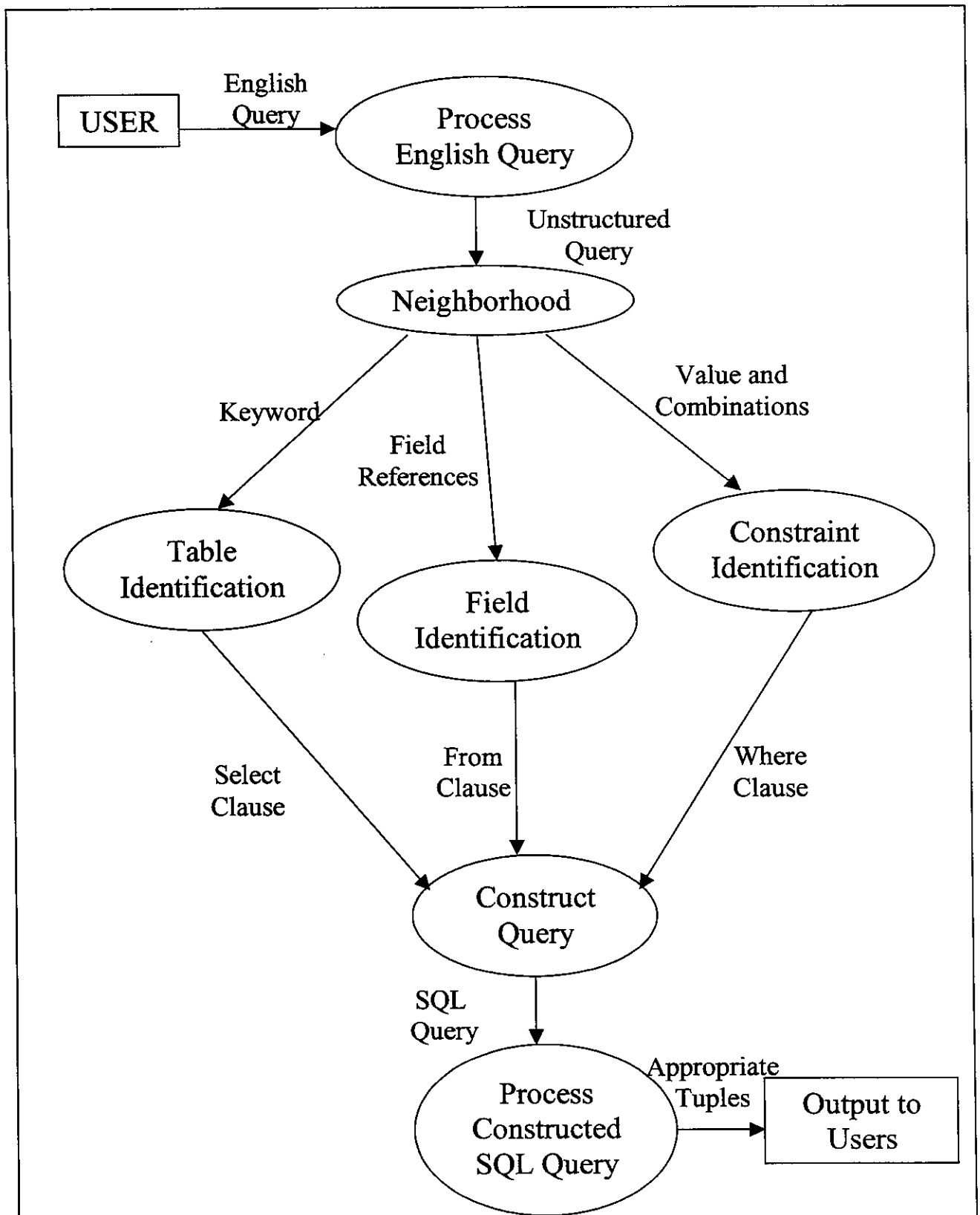


Fig 7.3 Level 1 DFD

SYSTEM DESIGN

8. SYSTEM DESIGN

8.1 MODULAR CLASSIFICATION

1. Neighborhood Module
2. Basic Identification Module - Identification of tables, fields and values
3. Combination Identification Module - Identify combinations (Related fields and values)
4. Query Rewriting Module - Rewriting the query and displaying the result
5. Intent Analysis Module- Provide links

8.2 IMPLEMENTATION TABLES

Table No. 8.1 ***Neighborhood***

Field	Type
Word	nvarchar
Synonym	nvarchar

Table No. 8.2 ***DataDictionary***

Field	Type
Keyword	nvarchar
TableName	nvarchar

Table No. 8.3 ***TableDetails***

Field	Type
TableName	nvarchar
FieldName	nvarchar
DefaultField	nvarchar

Table No. 8.4 ***Combination***

Field	Type
Operator	nvarchar
Opid	nvarchar

8.3 MODULES IN DETAIL

The presented query is first split into tokens.

8.3.1 Neighborhood Module

This module is incorporated to enhance the flexibility of the system. It provides the users with the freedom to use any word that is relevant to the domain in order to construct the query to retrieve the details required by him.

For this purpose we use a ***Neighborhood*** table as in TableNo.8.1. Every word in the query is compared with the entries in the ***Synonym*** field of the table and if there is a match then that word is replaced with the corresponding entry in the ***Word*** field of the table.

The query string at the end of this module will have only keywords, field references, values and/or comparison operators which can be used to directly identify fields and tables necessary to answer the query.

8.3.2 Basic Identification Module

The output of the neighborhood module is used to identify the tables and fields necessary to answer the query in the following three ways.

Identify the tables using keyword references

We use a ***DataDictionary*** table as in Table No.8.2. Each relevant word in the query is compared with the ***Keyword*** field in the table and if there is a match, the corresponding entry in the table is put in a dataset ***tabdet.*** This dataset will have the same fields that are there in the DataDictonary and the number of rows will be equal to the number of relevant keyword identified from the query string.

Identify the tables and fields using field references

We use a ***TableDetails*** table as in Table No.8.3. Each relevant word in the query that is not marked is compared with the ***FieldName*** field in the table and if there is a match, the corresponding entry is put in a dataset ***fielddet.*** This dataset will have the same fields as in the TableDetails and the number of rows will be equal to the number of relevant field references identified from the query string.

Identify tables and fields using value references

The value references in the query string are used to identify the fields and the corresponding tables that are needed to identify the tables needed to answer the query. Each relevant word in the query string that is not marked are passed to a function ***formhash( )*** each

table in the database. If the function finds the value in a particular table the function returns the fieldname of the table where the value has occurred. For every such fieldname returned we create a new row in a dataset ***hashvaltab***, using the table name, field name and the value reference identified.

8.3.3 Combination Identification Module

This module is aimed at handling the various comparison operators that can be used in a query so as to limit the display of irrelevant details to the user who presents the query.

For the purpose of handling such comparison operators in the query string we use a ***Combination*** table as in Table No.8.4 which has a list of all the possible comparison operators that can be used in the query string relevant to the domain of the system. We have also constructed a dataset ***combdet***. This dataset has ***Opname, Opid, Oppos, Opleft, Opright*** and ***Oprightl*** as its fields.

For each word that is unmarked in the query string, it is compared with the Operator field of the table and if there is a match then an entry is made in the ***combdet*** with the operator name and operator id in the ***Opname*** and ***Opid*** fields respectively. Then the position where the operator occurs in the query string is also entered in the ***Oppos*** field of the dataset as in Fig. no. ( )

The other three fields are used to make entries of the fields on which the operator operates and the values which are used to set up

constraints in the condition. In order to identify fields and values on which the operators operate we use the functions ***left ( ), right ( ), fright ( )*** and ***right2 ( ).*** These functions take the operator id and the position of the operator in the query string to do the processing. Their processing functionality is explained one by one in order in the following paragraphs.

The ***left ( )*** function is used to find the fields on which the operators operate. In any query the left side of the operator will be the field on which the operator is used. This function parses the query string, left from the position where the operator has occurred. Every relevant word is compared with the fields that are identified and entered in the dataset. As soon as it finds the nearest field left to the operator it makes an entry in the ***Opleft*** field of the dataset ***combdet.***

The ***right ( )*** and ***right2 ( )*** functions are used to find the values mentioned in the query string, using which the constraints are set on the respective fields. This functions similar to the ***left ( )***, only with a difference that it will parse the query string, right from the position where the operator occurred. The value identified first by ***right ( )*** is entered in the ***Opright*** field and the value identified second by the ***right2 ( )*** is entered in the ***Opright1*** field of ***combdet*** respectively.

The ***fright( )*** function is used to add more flexibility to the ***left ( )*** function. This performs the same functionality as that of the ***right ( )*** and ***right2 ( )*** function, except that it finds the first field immediately to the right of the operator, rather than the value. This is done only if the field on which the operator operates is unknown at the end of the execution of the above three functions.

In all the four mentioned functions, it is checked that if the value and the field on which the operator is used is compatible and whether the processing can be actually done.

Operators that can be handled

The various operators that can be handled are

i. >/greater/above
ii. </lesser/below
iii. =/equals
iv. first/best/top
v. second
vi. third
vii. last/worst/least
viii. various/distinct/different/list/enumerate
ix. between/range
x. not

These four datasets formed in module (2) and (3), namely ***tabdet***, ***fielddet***, ***hashvaltab*** and ***combdet*** form the basis of rewriting the English query into the SQL that has to be executed on the database in order to retrieve the necessary details.

8.3.4 Query Rewriting Module

This module uses the datasets, ***tabdet, fielddet, hashvaltab*** and ***combdet*** constructed in the above modules to actually restructure the English query presented by the user to the structured query that can be executed directly on the database in order to retrieve the details required by the user through the English query.

This module requires some preprocessing before using the tables, fields and values identified in the above mentioned datasets. Considering all the tables and fields identified in these datasets will cause redundant and unnecessary mentioning of the tables and fields in the rewritten query. In order to avert this situation, there is a need to consolidate the necessary fields and tables by comparing the entries in these datasets. The consolidation of tables and fields are done by creating two new datasets ***finaltablst*** and ***freshfielddet***.

The ***finaltablst*** has only one field ***TableName***. The final list of necessary tables is included in the ***finaltablst*** by making an entry for each entry in the ***tabdet*** and ***fielddet***.

The ***freshfielddet*** has two fields namely, ***FreshFname*** and ***FreshTname***. It is constructed in a similar manner by creating a new entry for every entry in the ***fielddet*** and the ***hashvaltab***.

Care is taken to remove redundant entries from both the datasets. This is done by deleting those entries in the datasets that exactly match a row that already exists.

Now is the actual query rewriting phase which is done three sections. Each section is devoted for a particular clause namely, the ***Select*** clause, ***From*** clause and the ***Where*** clause. These three clauses when put together will form the structured query which is required.

Select Clause

The parameters for the ***Select*** clause are a set of fields from which specific entries have to be retrieved. These set of fields are taken from ***freshfielddet*** and appended to complete the ***Select*** clause of the query.

From Clause

The parameters of the ***From*** clause are a set of tables form which specific rows have to be retrieved. These set of tables are taken from the ***finaltablst*** and appended to complete the ***From*** clause of the query.

Where Clause

The parameters of the ***Where*** clause are conditions or constraints. These constraints are of the form ***'FieldName op Value'***. If there are more than one constraints in the ***Where*** clause, these constraints are put together by conditional connectives. These constraints are formed by using the entries in the ***hashvaltab***. To incorporate conditional comparisons in the ***Where*** clause we use the entries in ***combdet***. Such comparative constraints are formed by using the appropriate constraint keyword such as between, distinct, orderby, count etc... This will complete the ***Where*** clause.

Then finally appending all the clauses formed with their respective parameters in the right order will give the fully structured query that is ready to be executed on the database to retrieve the required data.

8.3.5 Intent Analysis Module

This module is incorporated in the IQA system mainly to provide user friendliness. The aim of this module is to provide the user with the ability to choose from the several options that are available in the database relevant to his query. Such situations generally arise, mostly when the user's query is generalized or when the system has more data relevant to the query submitted, to provide to the user.

To provide this facility we have used the approach of redirecting the user to a new page, to provide him with the options that are available, so that the user can choose the options of his interest from there.

There are two important things that are to be considered before redirecting the user to the Selection page or providing a link to the extra information that can be provided by the system to the user. They are

- Keyword Ambiguity Problem (KAP)
- Field Ambiguity Problem (FAP)

Keyword Ambiguity Problem (KAP)

Keyword Ambiguity Problem means, that the same keyword in the ***DataDictionary*** maps to more than one table in the database. This can be identified in ***tabdet*** by checking if the same keyword entry has more than one table entry corresponding to it. If KAP is identified then steps have to be

taken to resolve this problem in order to answer the query correctly.

The way in which this can be resolved is as follows, ***tabdet*** entries have to be checked to see if there is an entry for one of the table entries corresponding to the ambiguous keyword. If such an entry exists, then it means that the KAP can be resolved by using that entry which has one of the table names of the ambiguous keyword but corresponding to another keyword. The other ambiguous entries of the keyword in ***tabdet*** can be deleted.

If such an entry does not exist and the above approach for resolving KAP cannot be used, then we redirect the user to another page to display all the tables corresponding to the ambiguous keyword so that the user can choose one table of his preference.

Field Ambiguity Problem(FAP)

Field Ambiguity Problem means, that the field with the same name is present in more than one table. This can be identified in ***fielddet*** when the same field entry has more than one table entry. . If FAP is identified then steps have to be taken to resolve this problem in order to answer the query correctly.

The way in which this problem can be resolved is as follows, ***tabdet*** entries have to be checked if there is any entry

for only one of the tables that corresponds to the ambiguous field name. If such an entry exists, then it means that the FAP can be resolved by using that entry, which has one of the table names of the ambiguous field. The other ambiguous entries of that field in ***fielddet*** can be deleted.

If such an entry does not exist and the above approach for resolving FAP cannot be used, then we redirect the user to another page to display all the tables corresponding to the ambiguous field so that the user can choose one table of his preference.

At times there is a possibility that both KAP and FAP exist and has to be resolved. In such a situation KAP is first solved using the above mentioned approach and then FAP, using the above mentioned approach.

Once the KAP and FAP are solved, the usual processing can carry on. Our next objective is to provide the users with not only details that can be retrieved by the exact matching method but also provide a chance for them to obtain extra information relevant to their query as and when it is possible. Whenever such a situation is identified, besides answering the query based on the exact matching method, the IQA system also provide the user with a link, clicking on which, the user is redirected to a new page that displays the available extra information the can be viewed by the user. The user can select the options according to his preference and view the other relevant particulars.

TESTING

9. TESTING

Testing is an important, mandatory part of software; it is a technique for evaluating product quality and also for indirectly improving it, by identifying defects and problems. It is required that the application should be tested for any non conformities and defects at every stage of development.

9.1 VALIDATION TESTING

We have tested our application at the end of every increment of the system. The IQA System being a user oriented one; utmost importance is given to validation testing. All possible queries that the user can pose, are considered and the respective validation is performed.

Validations Considered

- When user attempts to present an empty query, he is requested to type in the query and then proceed.
- The exceptions raised by the invalid queries posed by the user are handled by informing users about the mistake.
- When handling queries that involve the formation of conditions, missing parameters are identified and the users are informed about its absence.
- When the user requests for the tuples that are non-existent the user is informed about their non-existence.
- When a specified condition fails or there is no tuple that would match a complex condition, the user is let to know that the condition fails.

9.2 BLACK BOX TESTING

The black-box approach is a testing method in which the test data are derived from the specified functional requirements without regard to the final program structure. It is also termed data-driven, input/output driven or requirement based testing. All test cases are derived from the specification. No implementation details of the code are considered.

We have used the Rational Robot to generate the black-box test script for the IQA System developed.

Rational Robot automates regression, functional and configuration testing for e-commerce, client/server and ERP applications. It's used to test applications based upon a wide variety of user interface technologies, and is integrated with the Rational TestManager solution to provide desktop management support for all testing activities.

9.2.1 FEATURES OF RATIONAL ROBOT

- ***Simplifies configuration testing***

  Rational Robot can be used to distribute functional testing among many machines, each one configured differently. The same functional tests can be run simultaneously, shortening the time to identify problems with specific configurations.

- ***Tests many types of applications***

  Rational Robot supports a wide range of environments and languages, including HTML and DHTML, Java, VS.NET, Microsoft Visual Basic and Visual C++, Oracle Developer/2000, PeopleSoft, Sybase PowerBuilder and Borland Delphi.

- ***Ensures testing depth***

Tests beyond an application's UI to the hundreds of properties of an application's component objects - such as ActiveX Controls, OCXs, Java applets and many more - with just the click of a mouse.

- ***Tests custom controls and objects***

Rational Robot allows you to test each application component under varying conditions and provides test cases for menus, lists, alphanumeric characters, bitmaps and many more objects.

- ***Provides an integrated programming environment***

Rational Robot generates test scripts in SQABasic, an integrated MDI scripting environment that allows you to view and edit your test script while you are recording.

- ***Helps you analyze problems quickly***

Rational Robot automatically logs test results into the integrated Rational Repository, and color codes them for quick visual analysis. By double-clicking on an entry, you are brought directly to the corresponding line in your test script, thereby ensuring fast analysis and correction of test script errors.

- ***Enables reuse***

Rational Robot ensures that the same test script, without any modification, can be reused to test an application running on Microsoft Windows XP, Windows ME, Windows 2003, Windows 2000, Windows 98 or Windows NT

9.2.2 USE CASE SPECIFICATION

IntelligentQuery.aspx

Brief Description

This use case describes how an English query is converted to SQL query. The actor who use the use case are all those who use the system.

Flow of events

The use case begins when the actor enters the query

1. Actor should enter the query
2. If the actor clicks the 'Search' button the query is converted to sql query and the result is displayed.
3. If the actor clicks the 'NewQuery' button, the textbox is cleared and ready for a new query.

Alternative flows

1. If the actor clicks the search button when the textbox is empty, the system should display a message to the user asking him/her to enter a query.
2. If the actor types in an invalid query, the system should display an error message.
3. If the actor enters a wrong or incomplete query, hints are provided to help the user.
4. If the query is too general then the 'OtherDetails' button is highlighted.
5. When the actor clicks the 'OtherDetails' a new page is displayed which helps the user to select his/her specific options.

Pre conditions: Nil

Post Conditions: The result is displayed or the appropriate error message is displayed.

Selection.aspx

Brief Description

The use case describes how the user can select his preference.

Flow of events

The use case begins when the actor clicks details or gives generalized query.

Basic Flow

1. Actors should choose options from the list box.
2. When there are two list boxes the user can select multiple or all the options by clicking >> or 'All' button.
3. When the user clicks >> button, the selections are added one by one.
4. When the user clicks 'All' button, all the options are added.
5. When 'OK' button is clicked, the selections are passed to the main form.
6. When the actor clicks the 'Clear' button, the selections are cleared.

Alternative Flows

1. If the user selects an option more than once, a message is displayed to the user about the redundant selection.

Pre conditions

The system should identify that the query is generalized or the user has clicked the 'OtherDetails' button.

Post conditions

The selected option should be passed to the main page.

9.2.3 TEST SCRIPT

```
Sub Main
  Dim Result As Integer

  'Initially Recorded: 4/11/2003  3:10:54 AM
  'Script Name: iquerytest

  Window SetContext, "Caption=WebForm1 - Microsoft Internet Explorer", ""
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 1359
  EditBox Click, "Name=TextBox1", "Coords=224,29"
  SetThinkAvg 1906
  TypingDelays "0, 125, 156, 219, 296, 204, 234, 297, 78, 234, 625, 78, 157, 328, 1172"
  InputKeys "all those who a"
  SetThinkAvg 78
  TypingDelays "0, 187, 125, 391, 94, 140, 407, 93, 203, 235, 281, 94, 203, 375, 203,
156"
  InputKeys "re placed in CTS"
  SetThinkAvg 157
  TypingDelays "0, 219, 234, 157, 140, 781, 78, 187, 110, 343, 125, 204, 1203, 93, 532"
  InputKeys " and are in rot"
  SetThinkAvg 93
  TypingDelays "0, 282, 484, 94, 281, 281, 1141, 94, 390, 328, 2766, 891, 62, 188, 328"
  InputKeys "aract club in c"
  SetThinkAvg 344
  TypingDelays "0, 109"
  InputKeys "se"
  SetThinkAvg 1906
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 7922
  PushButton Click, "Name=Button5"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 1343
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 1453
  PushButton Click, "Name=Button5"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 1031
  EditBox Click, "Name=TextBox1", "Coords=83,27"
  SetThinkAvg 515
  TypingDelays "0, 16, 187, 16, 125"
  InputKeys "kfunj"
  SetThinkAvg 2578
  PushButton Click, "Name=Button1"
```

```
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 4313
PushButton Click, "Name=Button5"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 2969
EditBox Click, "Name=TextBox1", "Coords=74,32"
SetThinkAvg 3344
TypingDelays "0, 78, 94, 312, 32, 266, 344, 78"
InputKeys "students"
SetThinkAvg 1765
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 1328
PushButton Click, "Name=Button2"
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 1703
ListBox Click, "Name=ListBox2", "Text=City"
SetThinkAvg 719
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 578
ListBox VScrollTo, "Name=ListBox2", "Position=12"
SetThinkAvg 766
ListBox Click, "Name=ListBox2", "Text=State"
SetThinkAvg 797
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 1656
PushButton Click, "Name=Button3"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 906
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 5234
PushButton Click, "Name=Button5"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 844
EditBox Click, "Name=TextBox1", "Coords=86,35"
SetThinkAvg 5531
TypingDelays "0, 141, 531, 360, 78, 203, 78, 156, 625, 94, 562, 266, 875, 141, 265,
266"
InputKeys "semester1 > 80 a"
SetThinkAvg 93
TypingDelays "0, 219, 125, 375, 78, 453, 156, 235, 281, 140, 125, 344, 1031, 110,
265"
InputKeys "nd semester {BKSP}2 "
```

```
  SetThinkAvg 1250
  TypingDelays "0, 110, 219, 234, 234, 157, 343, 188, 2640, 79, 359, 187, 125, 172,
157"
  InputKeys "between 90 and "
  SetThinkAvg 953
  TypingDelays "0, 62, 938"
  InputKeys "95 "
  SetThinkAvg 2953
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm2",""
  SetThinkAvg 1047
  ListBox Click, "Name=ListBox2", "Text=Semester"
  SetThinkAvg 672
  PushButton Click, "Name=Button3"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 625
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 18938
  PushButton Click, "Name=Button5"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 1282
  EditBox Click, "Name=TextBox1", "Coords=134,37"
  SetThinkAvg 1812
  TypingDelays "0, 78, 125, 266, 94, 187, 1047, 625, 313, 78, 172, 453, 78, 141, 281,
156"
  InputKeys "studeb{BKSP}nts who a"
  SetThinkAvg 125
  TypingDelays "0, 109, 141, 437, 203, 235, 125, 343, 63, 109, 344, 78, 172"
  InputKeys "re not placed"
  SetThinkAvg 1578
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 2422
  PushButton Click, "Name=Button5"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 797
  EditBox Click, "Name=TextBox1", "Coords=124,46"
  SetThinkAvg 6188
  TypingDelays "0, 265, 797, 1313, 718, 157, 203, 265, 531, 47, 453, 235, 203, 281, 63"
  InputKeys "fis{BKSP}rst rank in"
  SetThinkAvg 203
  TypingDelays "0, 250, 203, 360, 390, 47, 297, 188, 172, 2140"
  InputKeys " monthly I"
  SetThinkAvg 1516
  PushButton Click, "Name=Button1"
```

```
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 891
ListBox Click, "Name=ListBox2", "Text=Mechanical"
SetThinkAvg 406
PushButton Click, "Name=Button3"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 797
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 3594
PushButton Click, "Name=Button2"
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 1985
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 1703
PushButton Click, "Name=Button2"
Browser NewPage,"HTMLTitle=WebForm2",""
SetThinkAvg 641
PushButton Click, "Name=Button3"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 1485
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 7938
PushButton Click, "Name=Button5"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 703
EditBox Click, "Name=TextBox1", "Coords=115,8"
SetThinkAvg 1188
TypingDelays "0, 78, 656, 78, 188"
InputKeys "abdul"
SetThinkAvg 1484
PushButton Click, "Name=Button1"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 3656
PushButton Click, "Name=Button3"
SetThinkAvg 1656
Toolbar Click, "ObjectIndex=4;\;ItemID=1014", "Coords=34,23"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 1484
PushButton Click, "Name=Button5"
Browser NewPage,"HTMLTitle=WebForm1",""
SetThinkAvg 688
EditBox Click, "Name=TextBox1", "Coords=46,23"
SetThinkAvg 1390
```

```
  TypingDelays "0, 78, 1203, 1891, 109"
  InputKeys "cse08"
  SetThinkAvg 1750
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm2",""
  SetThinkAvg 2328
  ListBox Click, "Name=ListBox2", "Text=ECActivities"
  SetThinkAvg 688
  PushButton Click, "Name=Button3"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 782
  PushButton Click, "Name=Button1"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 5281
  PushButton Click, "Name=Button5"
  Browser NewPage,"HTMLTitle=WebForm1",""
  SetThinkAvg 703
  EditBox Click, "Name=TextBox1", "Coords=132,23"

End Sub
```

10. FUTURE ENHANCEMENT

The IQA system developed so far has used the Query Rewriting Algorithm to convert English query to structured query. The system that has been developed uses the academic database of a particular class of students in a particular college to demonstrate its functionality.

10.1 Enhancement relevant to the domain considered

- The IQA system developed based on the Student domain, can be subjected to appreciable enhancement by extending its capability to answer queries not only based on a particular group of students but also based on the details of the students, staff, officials and other administrative details of an entire college or even an entire university.
- The system can be deployed in the intranet of the college or university, thereby facilitating easy and interactive retrieval of information to the management, staff and the wards of the students by giving the users their due privileges, thereby ensuring the security of the information stored in the database.

10.2 Enhancement in a broader sense

Very often, in any organization, there arises a need to view historical data. The task of searching and retrieving necessary data from large repositories becomes very tedious with the many procedures and formats that have to be followed in retrieving the required data. The IQA system can be used to make this task easy and interactive, by integrating it with the concept of Data Warehousing. The large repository of historical data of an organization can be built into a data warehouse and stored in the form of flat files which makes data retrieval easier.

CONCLUSION

11. CONCLUSION

The project entitled "***Intelligent Query Answering (IQA) System***" provides a well-developed, interactive GUI, which facilitates user-friendly querying. The proposed ***"Query Rewriting Algorithm"*** used to develop the system is very simple and comprehensive. Besides, the programming technique used in the design provides a greater scope of expansion for future use.

Invaluable experience has been gained in the areas of database access and query processing. The system has been implemented in such a way that it meets all the requirements gathered initially.

The IQA system has been checked and tested with a wide range of sample queries, covering most of the possible ways in which English queries can be posed. The project, being a user oriented one; utmost importance has been given to the validation phase in the development of the project.

The developed "Intelligent Query Answering (IQA) System" proves to be a useful solution for novice users of the respected domain who are not familiar with the Structured Querying Language.

APPENDIX

12. APPENDIX

12.1 DATABASE SCHEMA

Student Details

Field	Type
Student ID	nvarchar
Name	nvarchar
Address	nvarchar
City	nvarchar
State	nvarchar
Department	nvarchar
Course	nvarchar
X Aggregate	float
XII Aggregate	float
Entrance	Float
Payment Scheme	nvarchar

Attendance

Field	Type
Student ID	nvarchar
Semester1	float
Semester2	float
Semester3	float
Semester4	float
Semester5	float
Semester6	float
Semester7	float

Semester Average

Field	Type
Student ID	nvarchar
Semester1	float
Semester2	float
Semester3	float
Semester4	float
Semester5	float
Semester6	float
Semester7	float
Cumulative Average	float

Student Placements

Field	Type
Student ID	nvarchar
Company	nvarchar

Extra-Curricular Activities

Field	Type
Student ID	nvarchar
Category	nvarchar
Activity	nvarchar

Club Membership

Field	Type
Student ID	nvarchar
Club	nvarchar

Current Semester Marks (CSE/Mech/EEE/ECE/Civil)

Field	Type
Student ID	nvarchar
Monthly	nvarchar
Subject1	float
Subject2	float
Result	nvarchar
Total	float
Average	float
Rank	float

12.2 SAMPLE CODE

IntelligentQuery.aspx

```
    Private Sub Page_Load( ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
        conn = con.Connect
        Label2.Visible = False
        lblset = 0
        If Page.IsPostBack = False Then
            Dim nophoto = Session("exses")
            If nophoto = 1 Then
                Label2.Visible = True
                Label2.Text = "Photo not available"
                Dim w1 = Session("first")
                Dim w3 = " "
                Dim w2 = Session("selection")
                TextBox1.Text = w1 + w3 + w2
            ElseIf nophoto = 0 Then
                Dim w1 = Session("first")
                Dim w3 = " "
                Dim w2 = Session("selection")
                TextBox1.Text = w1 + w3 + w2
            End If
        End If
        Button2.Visible = False
        Button3.Visible = False
    End Sub

'Query Splitting
dupstr = TextBox1.Text
str = TextBox1.Text.Trim.ToLower.Split

  Public Function neighbor()
        Dim i As Int16, newc As Int16, oldc As Int16, l As Int16
        Dim oldword As String, newword As String, a() As String
        Dim neighcmd As SqlCommand
        Dim neighadap As SqlDataAdapter
        Dim neighdet As DataTable
        neighdet = New DataTable

        i = str.Length
        Do While i
            Dim a1 As String
            a1 = str(i - 1)
            If intstr(i - 1) <> 1 Then
                oldc = neighdet.Rows.Count
                neighcmd = New SqlCommand("SELECT *
                FROM Neighborhood WHERE Synonym
                LIKE '" & str(i - 1) & "'", conn)
                neighadap = New SqlDataAdapter(neighcmd)
                neighadap.Fill(neighdet)
                newc = neighdet.Rows.Count
                If newc > oldc Then
                    oldword = str(i - 1)
```

```
                    newword = neighdet.Rows(neighdet.Rows.Count - 1)(0)
                    str(i - 1) = newword
                End If
            End If
            i = i - 1
        Loop

    End Function

    'Identifying keywords and the corresponding table
    Dim tcmd As SqlCommand, tadap As SqlDataAdapter
    i = str.Length
    tabdet = New DataTable
    Do While i
        If intstr(i - 1) <> 1 Then
                tcmd = New SqlCommand("SELECT distinct *
                FROM  DataDictionary WHERE Keyword LIKE
                '" & str(i - 1) & "'", conn)
                tadap = New SqlDataAdapter(tcmd)
                tadap.Fill(tabdet)
        End If
        i = i - 1
    Loop

'Identifying fields and the corresponding tables
    Dim fcmd As SqlCommand, fadap As SqlDataAdapter
    fielddet = New DataTable
    For j = 0 To str.Length - 1
        If intstr(j) <> 1 Then
            fcmd = New SqlCommand("SELECT * FROM
            TableDetails WHERE FieldName LIKE
            '" & str(j) & "'", conn)
            fadap = New SqlDataAdapter(fcmd)
            fadap.Fill(fielddet)
        End If
    Next

'Pass the different tables to the formhash function
    Dim columnname As String, tn As String, isnum As Int16
    Dim hvtr As DataRow
    hashvaltab = New DataTable
    Dim Value As DataColumn = New DataColumn("Value")
    Value.DataType = System.Type.GetType("System.String")
    hashvaltab.Columns.Add(Value)
    Dim ColNam As DataColumn = New DataColumn("ColNam")
    ColNam.DataType = System.Type.GetType("System.String")
    hashvaltab.Columns.Add(ColNam)
    Dim TablName As DataColumn = New DataColumn("TablName")
    TablName.DataType = System.Type.GetType("System.String")
    hashvaltab.Columns.Add(TablName)

    i = str.Length
    Do While i
        If intstr(i - 1) <> 1 Then
            Dim reqvalue As String
            reqvalue = str(i - 1)
            isnum = 0
```

```
            Dim z As Int16
            For z = 0 To numtab.Rows.Count - 1
                Dim templ As String
                templ = numtab.Rows(z)(0)
                If reqvalue.StartsWith(templ) Then
                    isnum = 1
                End If
            Next

            If isnum = 0 Then
                foundval = 0
                For j = 0 To tables.Rows.Count - 1
                    tn = tables.Rows(j)(0)
                    columnname = formhash(tn, reqvalue)
                    If foundval = 1 Then
                        Dim ambsql As SqlCommand,
                        defsql As SqlCommand,
                        adadap As SqlDataAdapter
                        Dim amb As String, def As String
                        ambsql = New SqlCommand("Select *
                        from TableDetails where
                        Tablename='" & tn & "' and
                        FieldName='" & columnname & "'", conn)
                        adadap = New SqlDataAdapter(ambsql)
                        adadap.Fill(fielddet)
                        hvtr = hashvaltab.NewRow()
                        hvtr.Item("Value") = reqvalue
                        hvtr.Item("ColNam") = columnname
                        hvtr.Item("TablName") = tn
                        hashvaltab.Rows.Add(hvtr)
                    End If
                Next
            End If
        End If
        i = i - 1
    Loop

'Final identification of the required tables with the above two
identification
'Forming a new final table list from tabdet and fielddet
Dim tr As DataRow
finaltablst = New DataTable("TableList")
Dim TableName As DataColumn = New DataColumn("TableName")
TableName.DataType = System.Type.GetType("System.String")
finaltablst.Columns.Add(TableName)

For i = 0 To tabdet.Rows.Count - 1
    tr = finaltablst.NewRow()
    tr.Item("TableName") = tabdet.Rows(i)(1)
    finaltablst.Rows.Add(tr)
Next
For i = 0 To fielddet.Rows.Count - 1
    tr = finaltablst.NewRow()
    tr.Item("TableName") = fielddet.Rows(i)(0)
    finaltablst.Rows.Add(tr)
Next
```

```
'To form freshfielddet
Dim ffr As DataRow
freshfdet = New DataTable("FreshFieldList")
Dim FreshFName As DataColumn = New DataColumn("FreshFName")
FreshFName.DataType = System.Type.GetType("System.String")
freshfdet.Columns.Add(FreshFName)
Dim FreshTName As DataColumn = New DataColumn("FreshTName")
 FreshTName.DataType = System.Type.GetType("System.String")
 freshfdet.Columns.Add(FreshTName)

 For i = 0 To fielddet.Rows.Count - 1
      ffr = freshfdet.NewRow()
      ffr.Item("FreshFName") = fielddet.Rows(i)(1)
      ffr.Item("FreshTName") = fielddet.Rows(i)(0)
      freshfdet.Rows.Add(ffr)
 Next
 For i = 0 To hashvaltab.Rows.Count - 1
      ffr = freshfdet.NewRow()
      ffr.Item("FreshFName") = hashvaltab.Rows(i)(1)
      ffr.Item("FreshTName") = hashvaltab.Rows(i)(2)
      freshfdet.Rows.Add(ffr)
 Next

'Forming the final query
 fstr = New StringBuilder

'Select Clause
fstr.Append("select
StudentDetails.StudentID,StudentDetails.StudentName")
For i = 0 To freshfdet.Rows.Count - 1
      Dim fn As String
      fn = freshfdet.Rows(i)(0)
      If fn <> "StudentID" And fn <> "StudentName" Then
            fstr.Append(",")
            fstr.Append(fn)
      End If
 Next

'From Clause
 fstr.Append(" from ")
 For i = 0 To finaltablst.Rows.Count - 1
      Dim tnl As String
      tnl = finaltablst.Rows(i)(0)
      fstr.Append(tnl)
      fstr.Append(",")
 Next
 l = fstr.Length
 fstr.Remove(l - 1, 1)

'Where Clause
If hashvaltab.Rows.Count > 0 Then
      If finaltablst.Rows.Count = 1 Then
            fstr.Append(" where ")
      End If
End If
```

```
    interstr.Append(" where ")
    For i = 0 To finaltablst.Rows.Count - 2
           j = i + 1
           If i = finaltablst.Rows.Count - 2 Then
                 j = 0
           End If
           interstr.Append(finaltablst.Rows(i)(0)
           + ".StudentID=")
           interstr.Append(finaltablst.Rows(j)(0)
           + ".StudentID")
           If i <> finaltablst.Rows.Count - 2 Then
                 interstr.Append(" and ")
           End If
    Next
Dim val As String
f = 0
If hashvaltab.Rows.Count > 0 Then
      For i = 0 To hashvaltab.Rows.Count - 1
           If finaltablst.Rows.Count > 1 Then
                 fstr.Append(" and ")
           End If
           fieldname = hashvaltab.Rows(i)(1)
           s = "StudentID"
           If s.Equals(fieldname) Then
                 fstr.Append("StudentDetails.")
           End If
           fstr.Append(hashvaltab.Rows(i)(1))
           fstr.Append(" = ")
           val = hashvaltab.Rows(i)(0)
           fstr.Append("'" & val.ToString & "'")
            If finaltablst.Rows.Count = 1 Then
                 fstr.Append(" and ")
                 f = 1
            End If
         Next
 End If
 If f = 1 Then
       l = fstr.Length
       fstr.Remove(l - 4, 4)
 End If

 Dim finaladap As SqlDataAdapter
 Dim finalcmd As SqlCommand
 finaltab = New DataTable

 'Displaying the answer
 finalcmd = New SqlCommand("" & fstr.ToString & "", conn)
 finaladap = New SqlDataAdapter(finalcmd)
 finaladap.Fill(finaltab)
 finaltab = remredundantrows(finaltab)
```

```
'Combinations
 combitab = New DataTable
 Dim combtab As DataTable
 Dim combcmd As New SqlCommand
 Dim combadap As New SqlDataAdapter
 combdet = New DataTable
 combtab = New DataTable
 combdet = New DataTable("combdet")
 Dim Opname As DataColumn = New DataColumn("Opname")
 Opname.DataType = System.Type.GetType("System.String")
 combdet.Columns.Add(Opname)
 Dim Opid As DataColumn = New DataColumn("Opid")
 Opid.DataType = System.Type.GetType("System.String")
 combdet.Columns.Add(Opid)
 Dim Oppos As DataColumn = New DataColumn("Oppos")
 Oppos.DataType = System.Type.GetType("System.String")
 combdet.Columns.Add(Oppos)
 Dim Opleft As DataColumn = New DataColumn("Opleft")
 Opleft.DataType = System.Type.GetType("System.String")
 combdet.Columns.Add(Opleft)
 Dim Opright As DataColumn = New DataColumn("Opright")
 Opright.DataType = System.Type.GetType("System.String")
 combdet.Columns.Add(Opright)
 Dim Opright1 As DataColumn = New DataColumn("Oprightl")
 Opright1.DataType = System.Type.GetType("System.String")
 combdet.Columns.Add(Oprightl)

 'Identifying combinations
 Dim i As Int16 = 0, oldcnt As Int16 = 0, newcnt As Int16 = 0
 Dim j As Int16 = 0
 Dim op As String
 Dim cr As DataRow


 i = str.Length
      Do While i
            op = str(i - 1)
            oldcnt = combtab.Rows.Count
            combcmd = New SqlCommand("SELECT * FROM Combination WHERE
            Operator LIKE '" & str(i - 1) & "'", conn)
            combadap = New SqlDataAdapter(combcmd)
            combadap.Fill(combtab)
            newcnt = combtab.Rows.Count
            If newcnt > oldcnt Then
                cr = combdet.NewRow()
                cr.Item("Opname") = combtab.Rows(j)(0)
                cr.Item("Opid") = combtab.Rows(j)(1)
                cr.Item("Oppos") = i - 1
                combdet.Rows.Add(cr)
                j = j + 1
            End If
            i = i - 1
        Loop

 Public Function left(ByVal cdt As DataTable, ByVal str() As String,
 ByVal rn As Int16)
```

```
Dim i As Int16, op As Int16, j As Int16, k As Int16
Dim w1 As String, w2 As String, zword As String
Dim temp As String, w As String
Dim entry As Int16 = 0, opident As Int16 = 0, num As Int16 = 0

opident = cdt.Rows(rn)(1)
If opident = 1 Or opident = 2 Or opident = 3 Then
    num = 1
End If
temp = cdt.Rows(rn)(2)
op = CType(temp, Int16)
entry = 0

Do While op + 1
    For k = 0 To freshfdet.Rows.Count - 1
        w = str(op)
        w1 = freshfdet.Rows(k)(0)
        w2 = freshfdet.Rows(k)(1)
        If w.Equals(w1.ToLower) Then
            If num = 1 Then
                Dim zcmd As SqlCommand
                Dim zadap As SqlDataAdapter, ztab As DataTable
                ztab = New DataTable
                zcmd = New SqlCommand("SELECT " & w1 & " from "
                & w2 & "", conn)
                Dim q As String
                q = "SELECT '" & w1 & "' from '" & w2 & "'"
                zadap = New SqlDataAdapter(zcmd)
                zadap.Fill(ztab)
                zword = ztab.Rows(0)(0)
                Dim isnum As Int16, temp1 As String, z As Int16
                isnum = 0

                For z = 0 To numtab.Rows.Count - 1
                    temp1 = numtab.Rows(z)(0)
                    If zword.StartsWith(temp1) Then
                        isnum = 1
                    End If
                Next
                If isnum = 1 Then
                    cdt.Rows(rn)(3) = str(op)
                    entry = 1
                    If entry = 1 Then
                        Exit For
                    End If
                End If
            ElseIf num = 0 Then
                cdt.Rows(rn)(3) = str(op)
                entry = 1
                If entry = 1 Then
                    Exit For
                End If
            End If
        End If
    Next
    If entry = 1 Then
```

```
            Exit Do
        End If
        op = op - 1
    Loop

    If entry = 0 Then
        cdt = fright(cdt, str, rn)
    End If

    Return cdt
End Function

Public Function right(ByVal cdt As DataTable,
ByVal str() As      String, ByVal rn As Int16)

    Dim i As Int16, j As Int16, op As Int16, entry As Int16 = 0
    Dim temp As String, temp1 As String, w As String

    temp = cdt.Rows(rn)(2)
    op = CType(temp, Int16)
    For i = op To str.Length - 1
        Response.Write(str.Length)
        w = str(i)
        For j = 0 To numtab.Rows.Count - 1
            temp1 = numtab.Rows(j)(0)
            If w.StartsWith(temp1) Then
                cdt.Rows(rn)(4) = str(i)
                entry = 1
                newi = i + 1
                If entry = 1 Then
                    Exit For
                End If
            End If
        Next
        If entry = 1 Then
            Exit For
        End If
    Next
    Return cdt
End Function

Public Function combgle(ByVal cdt As DataTable,
ByVal str() As String, ByVal rn As Int16, ByVal id As Int16)

    Dim val As String
    Dim i As Int16, l As Int16, f As Int16

    Try
        f = 0
        If finaltablst.Rows.Count > 1 Then
            combstr.Append(" and ")
        End If
        Try
            Dim s1 As String
            s1 = cdt.Rows(rn)(3)
            combstr.Append(cdt.Rows(rn)(3))
        Catch ex As Exception
```

```
            Label2.Visible = True
            Label2.Text = "Criteria name expected" + " " +
            cdt.Rows(rn)(0) + " " + cdt.Rows(rn)(4)
            lblset = 1
            dg.Visible = False
            exflag = 1
            Exit Function
        End Try

        If id = "1" Then
            combstr.Append(" > ")
        ElseIf id = "2" Then
            combstr.Append(" < ")
        ElseIf id = "3" Then
            combstr.Append(" = ")
        End If

        Try
            val = cdt.Rows(rn)(4)
        Catch ex As Exception
            Label2.Visible = True
            Label2.Text = cdt.Rows(rn)(3) + " " + cdt.Rows(rn)(0) +
            " " + "Value expected within 100"
            lblset = 1
            dg.Visible = False
            exflag = 1
            Exit Function
        End Try

        combstr.Append("'" & val.ToString & "'")
        If finaltablst.Rows.Count = 1 Then
            combstr.Append(" and ")
            f = 1
        End If
        If f = 1 Then
            l = combstr.Length
            combstr.Remove(l - 4, 4)
        End If
    Catch ex As Exception
        Label2.Visible = True
        Label2.Text = "Criteria name" + " " + cdt.Rows(rn)(0) + " "
        + "Value within 100"
        lblset = 1
        dg.Visible = False
        exflag = 1
        Exit Function
    End Try

End Function

Public Function fright(ByVal cdt As DataTable,
ByVal str() As String, ByVal rn As Int16)

    Dim i As Int16, j As Int16, k As Int16, op As Int16
    Dim temp As String, w As String, zword As String, w1 As String,
    Dim w2 As String
    Dim entry As Int16 = 0, opident As Int16 = 0, num As Int16 = 0
```

```
    opident = cdt.Rows(rn)(1)
    If opident = 1 Or opident = 2 Or opident = 3 Then
        num = 1
    End If
    temp = cdt.Rows(rn)(2)
    op = CType(temp, Int16)
    entry = 0

    For i = op To str.Length - 1
        For k = 0 To freshfdet.Rows.Count - 1
            w = str(i)
            w1 = freshfdet.Rows(k)(0)
            w2 = freshfdet.Rows(k)(1)
            If w.Equals(w1.ToLower) Then
                If num = 1 Then
                    Dim zcmd As SqlCommand
                    Dim zadap As SqlDataAdapter, ztab As DataTable
                    ztab = New DataTable
                    zcmd = New SqlCommand("SELECT " & w1 & " from "
                    & w2 & "", conn)
                    Dim q As String
                    q = "SELECT '" & w1 & "' from '" & w2 & "'"
                    zadap = New SqlDataAdapter(zcmd)
                    zadap.Fill(ztab)
                    zword = ztab.Rows(0)(0)
                    Dim isnum As Int16, temp1 As String, z As Int16
                    isnum = 0
                    For z = 0 To numtab.Rows.Count - 1
                        temp1 = numtab.Rows(z)(0)
                        If zword.StartsWith(temp1) Then
                            isnum = 1
                        End If
                    Next
                    If isnum = 1 Then
                        cdt.Rows(rn)(3) = str(i)
                        entry = 1
                        If entry = 1 Then
                            Exit For
                        End If
                    End If
                ElseIf num = 0 Then
                    cdt.Rows(rn)(3) = str(i)
                    entry = 1
                    If entry = 1 Then
                        Exit For
                    End If
                End If
            End If
        Next
        If entry = 1 Then
            Exit For
        End If
    Next

    Return cdt
End Function
```

```
Public Function right2( ByVal cdt As DataTable,
ByVal str( ) As String, ByVal rn As Int16)

    Dim i As Int16, j As Int16, op As Int16, entry As Int16 = 0
    Dim temp As String, temp1 As String, w As String

    temp = cdt.Rows( rn)( 2)
    op = CType( temp, Int16)
    For i = newi To str.Length - 1
        Response.Write( str.Length)
        w = str( i)
        For j = 0 To numtab.Rows.Count - 1
            temp1 = numtab.Rows( j)( 0)
            If w.StartsWith( temp1) Then
                cdt.Rows( rn)( 5) = str( i)
                entry = 1
                If entry = 1 Then
                    Exit For
                End If
            End If
        Next
        If entry = 1 Then
            Exit For
        End If
    Next

    Return cdt
End Function
```

```
Public Function comborder( ByVal cdt As DataTable,
ByVal str( ) As String, ByVal rn As Int16, ByVal id As Int16)

    Dim val As String, field As String, w As String
    Dim i As Int16, 1 As Int16
    Dim combocmd As SqlCommand
    Dim comboadap As SqlDataAdapter
    Dim combotab As DataTable
    Dim f As Int16

    Try
        f = 0
        field = cdt.Rows( rn)( 3)
        If field.Equals( "rank") Then
            combstr.Append( " and rank! ='0' order by monthly,rank ")
        Else
            combstr.Append( " order by ")
            combstr.Append( field)
            combstr.Append( " desc ")
        End If
        Dim combquery As String
        combquery = "" & combstr.ToString & ""
        combotab = New DataTable
```

```
    combocmd = New SqlCommand( combquery.ToString, conn)
    comboadap = New SqlDataAdapter( combocmd)
    comboadap.Fill( combotab)
Catch ex As Exception
    Label2.Visible = True
    Label2.Text = cdt.Rows( rn)( 0) + " in 'Criteria Name'
    rank/monthly I/monthly II/subject1/subject2/semester"
    lblset = 1
    dg.Visible = False
    exflag = 1
    Exit Function
End Try

Dim noc As Int16
onelinetab = New DataTable
Dim FieldName As DataColumn = New DataColumn( "FieldName")
FieldName.DataType = System.Type.GetType( "System.String")
onelinetab.Columns.Add( FieldName)
Dim Value As DataColumn = New DataColumn( "Value")
Value.DataType = System.Type.GetType( "System.String")
onelinetab.Columns.Add( Value)
noc = combotab.Columns.Count
Dim cr As DataRow

For i = 0 To noc - 1
    cr = onelinetab.NewRow( )
    cr.Item( "FieldName") = combotab.Columns( i).ColumnName
    onelinetab.Rows.Add( cr)
Next



If id = "4" Then
    For i = 0 To noc - 1
        onelinetab.Rows( i)( 1) = combotab.Rows( 0)( i)
    Next
    dg.DataSource = onelinetab
    dg.DataBind( )
    Button3.Visible = True
    p = 1
    Label5.Text = "" & combstr.ToString & ""
End If
If id = "5" Then
    For i = 0 To noc - 1
        onelinetab.Rows( i)( 1) = combotab.Rows( 1)( i)
    Next
    dg.DataSource = onelinetab
    dg.DataBind( )
    Button3.Visible = True
    p = 1
    Label5.Text = "" & combstr.ToString & ""
End If
If id = "6" Then
    For i = 0 To noc - 1
        onelinetab.Rows( i)( 1) = combotab.Rows( 2)( i)
```

```
        Next
        dg.DataSource = onelinetab
        dg.DataBind()
        Button3.Visible = True
        p = 1
        Label5.Text = "" & combstr.ToString & ""
    End If
    If id = "7" Then
        For i = 0 To noc - 1
            onelinetab.Rows(i)(1) =
            combotab.Rows(combotab.Rows.Count - 1)(i)
        Next
        dg.DataSource = onelinetab
        dg.DataBind()
        Button3.Visible = True
        p = 1
        Label5.Text = "" & combstr.ToString & ""
    End If
End Function

Public Function combvarious(ByVal cdt As DataTable,
ByVal str() As String, ByVal rn As Int16, ByVal id As Int16)

    Dim varstr As StringBuilder
    Dim field As String, varquery As String, tname As String,
    Dim field1 As String, vartab As DataTable
    Dim i As Int16

    Try
        varstr = New StringBuilder
        varstr.Append("Select distinct ")
        field = cdt.Rows(rn)(3)
        varstr.Append(field)
        varstr.Append(" from ")
        Dim n As Int16
        n = fielddet.Rows.Count
        For i = 0 To fielddet.Rows.Count - 1
            field1 = fielddet.Rows(i)(1)
            If field.Equals(field1.ToLower) Then
                tname = fielddet.Rows(i)(0)
                varstr.Append(tname)
                Exit For
            End If
        Next
        Dim varcmd As SqlCommand
        Dim varadap As SqlDataAdapter
        vartab = New DataTable
        varcmd = New SqlCommand(varquery, conn)
        varadap = New SqlDataAdapter(varcmd)
        varadap.Fill(vartab)
    Catch ex As Exception
        i = str.Length
        Do While i
            Dim w As String
            w = str(i - 1)
            If w.Equals("student") Then
                Exit Function
```

```
            End If
            i = i - 1
        Loop
        Label2.Visible = True
        Label2.Text = "Criteria can be
        Departments/Companies/Clubs/Activities/Categories"
        lblset = 1
        dg.Visible = False
        exflag = 1
        Exit Function
    End Try

    If vartab.Rows.Count > 0 Then
        dg.DataSource = vartab
        dg.DataBind()
        p = 1
        Label5.Text = varquery.ToString
    End If
End Function

Public Function combnumber()
    Dim num As Int16
    Dim numdt As DataTable
    Dim numr As DataRow
    numdt = New DataTable
    Dim Number As DataColumn = New DataColumn("Number")
    Number.DataType = System.Type.GetType("System.String")
    numdt.Columns.Add(Number)
    num = finaltab.Rows.Count
    numr = numdt.NewRow()
    numr.Item("Number") = num.ToString
    numdt.Rows.Add(numr)
    If numdt.Rows.Count > 0 Then
        dg.DataSource = numdt
        dg.DataBind()
        p = 1
    End If
End Function
Public Function combbet(ByVal cdt As DataTable,
ByVal str() As String, ByVal rn As Int16)
    Dim val As String
    Dim i As Int16, l As Int16, f As Int16

    f = 0
    If finaltablst.Rows.Count > 1 Then
        combstr.Append(" and ")
    End If

    Try
        Try
            combstr.Append(cdt.Rows(rn)(3))
        Catch ex As Exception
            Label2.Visible = True
            Label2.Text = "Criteria Name Expected"
            lblset = 1
            dg.Visible = False
            exflag = 1
```

```
                Exit Function
            End Try
            combstr.Append(" between ")
            val = cdt.Rows(rn)(4)
            combstr.Append("'" & val.ToString & "'")
            combstr.Append(" and ")
            val = cdt.Rows(rn)(5)
            combstr.Append("'" & val.ToString & "'")
            If finaltablst.Rows.Count = 1 Then
                combstr.Append(" and ")
                f = 1
            End If

            If f = 1 Then
                l = combstr.Length
                combstr.Remove(l - 4, 4)
            End If
        Catch ex As Exception
            Label2.Visible = True
            Label2.Text = "Criteria BETWEEN value and value (value
            within 100)"
            lblset = 1
            dg.Visible = False
            exflag = 1
            Exit Function
        End Try

    End Function

    Public Function combnot()

        Dim nottab As DataTable
        nottab = New DataTable
        Dim notcmd As SqlCommand, notadap As SqlDataAdapter
        notcmd = New SqlCommand("SELECT StudentID,StudentName from
        StudentDetails", conn)
        notadap = New SqlDataAdapter(notcmd)
        notadap.Fill(nottab)
        If finaltab.Rows.Count = 0 Then
            Label2.Visible = True
            Label2.Text = "No one matches the criteria specified"
            lblset = 1
            dg.Visible = False
            exflag = 1
            Exit Function
        End If
lab:    Dim i As Int16, del As Int16 = 0, j As Int16
        For i = 0 To nottab.Rows.Count - 1
            del = 0
            Dim roll1 As String
            roll1 = nottab.Rows(i)(0)
            For j = 0 To finaltab.Rows.Count - 1
                Dim roll2 As String
                roll2 = finaltab.Rows(j)(0)
                If roll1.Equals(roll2) Then
                    nottab.AcceptChanges()
                    nottab.Rows(i).Delete()
```

```
                nottab.AcceptChanges()
                del = 1
                Exit For
            End If
        Next
        If del = 1 Then
            Exit For
        End If
    Next
    If del = 1 Then
        GoTo lab
    End If
    If nottab.Rows.Count > 0 Then
        dg.DataSource = nottab
        dg.DataBind()
        p = 1
    End If
End Function

Public Function links()
    Dim linkcmd As SqlCommand, linkadap As SqlDataAdapter,
    Dim linkdet As DataTable, tname As String, i As Int16
    Dim s As String
    linkdet = New DataTable
    If tabdet.Rows.Count = 0 And fielddet.Rows.Count = 0 Then
        If lblset = 0 Then
            Label2.Visible = True
            Label2.Text = "Click OTHER DETAILS to retrieve the
            necessary data"
            lblset = 1
            Button2.Visible = True
            TextBox1.Enabled = False
            Session("sesint") = 1
        End If
    End If

    'If only a students name or id is specified
    If tabdet.Rows.Count = 0 And fielddet.Rows.Count = 2 Then
        Dim fld As String = fielddet.Rows(0)(1)
        Dim fld1 As String = fielddet.Rows(1)(1)
        If fld.Equals("StudentName") And fld1.Equals("StudentID")
        Then
            If lblset = 0 Then
                Label2.Visible = True
                Label2.Text = "Click OTHER DETAILS to retrieve the
                necessary data"
                lblset = 1
                Button2.Visible = True
                TextBox1.Enabled = False
                Session("sesint") = 1
            End If
        End If
    End If
    Dim valflag As Int16 = 0
    For i = 0 To hashvaltab.Rows.Count - 1
        Dim t1 As String, t2 As String
        t1 = hashvaltab.Rows(i)(0)
```

```
        Dim j As Int16
        For j = 0 To tabdet.Rows.Count - 1
            t2 = tabdet.Rows(j)(1)
            If t1.Equals(t2.ToLower) Then
                If lblset = 0 Then
                    Label2.Visible = True
                    Label2.Text = "Click OTHER DETAILS to retrieve
                    the necessary data"
                    lblset = 1
                    Button2.Visible = True
                    TextBox1.Enabled = False
                    valflag = 1
                    Session("sesint") = 4
                    Session("ses") = t2
                    Exit For
                End If
            End If
        Next
        If valflag = 1 Then
            Exit For
        End If
    Next

   If combitab.Columns.Count = 2 Then
        If lblset = 0 Then
            Label2.Visible = True
            Label2.Text = "Click OTHER DETAILS to retrieve the
            necessary data"
            lblset = 1
            Button2.Visible = True
            TextBox1.Enabled = False
            Session("sesint") = 1
        End If
    End If
End Function

Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
    dupstr = TextBox1.Text
    If Session("sesint") = 4 Then
        Session("first") = dupstr
        Response.Redirect("selection.aspx")
    Else
        Session("sesint") = 1
        Session("ses") = dupstr
        Session("first") = dupstr
        Response.Redirect("selection.aspx")
    End If
End Sub
```

Selection.aspx

```
Private Sub Page_Load( ByVal sender As System. Object,
ByVal e As System. EventArgs) Handles MyBase. Load
    conn = con. Connect
    ListBox1. Visible = True
    Button1. Visible = True
    Dim dint = Session( "sesint")
    Dim d = Session( "ses")
    s = Session( "first")
    If Page. IsPostBack = False Then
        If dint = 1 Then
            Dim sql1 As StringBuilder
            Dim sqlremtab As String, adapremtab As SqlDataAdapter
            Dim dt2 As DataTable
            sqlremtab = "select TableName from TableDetails where
            DefaultField='2' "
            adapremtab = New SqlDataAdapter( sqlremtab, conn)
            dt2 = New DataTable
            adapremtab. Fill( dt2)
            sql1 = New StringBuilder
            sql1. Append( "select distinct FieldName from
            TableDetails where FieldName != 'StudentID' and
            FieldName! ='StudentName' ")
            Dim i As Int16
            For i = 0 To dt2. Rows. Count - 1
                sql1. Append( " and TableName! =' " +
                dt2. Rows( i)( 0) + "' ")
            Next
            Dim dt1 As DataTable
            adap = New SqlDataAdapter( sql1. ToString, conn)
            dt1 = New DataTable
            adap. Fill( dt1)
            ListBox2. DataSource = dt1
            ListBox2. DataTextField = "FieldName"
            ListBox2. DataBind( )
        ElseIf dint = 0 Then
            ListBox2. DataSource = d
            ListBox2. DataTextField = "TableN"
            ListBox2. DataBind( )
            ListBox1. Visible = False
            Button1. Visible = False
            Button2. Visible = False
            Button4. Visible = False
        ElseIf dint = 2 Then
            Dim dt As New DataTable
            Dim sql = "select distinct TableName from
            DataDictionary where Keyword=' " & d & "' "
            adap = New SqlDataAdapter( sql, conn)
            adap. Fill( dt)
            ListBox2. DataSource = dt
            ListBox2. DataTextField = "TableName"
            ListBox2. DataBind( )
            ListBox1. Visible = False
            Button1. Visible = False
            Button2. Visible = False
```

```
            Button4.Visible = False
        ElseIf dint = 3 Then
            Dim sql = "select FieldName from TableDetails where
            TableName='" & d & "' and DefaultField='1' and
            FieldName != 'StudentID' and FieldName!='StudentName' "
            adap = New SqlDataAdapter(sql, conn)
            Dim dt As New DataTable
            adap.Fill(dt)
            ListBox2.DataSource = dt
            ListBox2.DataTextField = "FieldName"
            ListBox2.DataBind()
        ElseIf dint = 4 Then
            Dim sql = "select FieldName from TableDetails where
            TableName='" & d & "' and FieldName != 'StudentID' and
            FieldName!='StudentName' "
            adap = New SqlDataAdapter(sql, conn)
            Dim dt As New DataTable
            adap.Fill(dt)
            ListBox2.DataSource = dt
            ListBox2.DataTextField = "FieldName"
            ListBox2.DataBind()
        ElseIf dint = 5 Then
            ListBox2.DataSource = d
            ListBox2.DataTextField = "Ambtab"
            ListBox2.DataBind()
            ListBox1.Visible = False
            Button1.Visible = False
            Button2.Visible = False
            Button4.Visible = False
        End If
    End If
End Sub

Private Sub Button3_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button3.Click
    Dim ss As StringBuilder
    ss = New StringBuilder
    Dim i As Int16
    For i = 0 To ListBox1.Items.Count - 1
        Dim st1 As String
        st1 = ListBox1.Items(i).ToString
        ss.Append(" ")
        ss.Append(st1)
    Next
    Session("selection") = ss.ToString
    Session("first") = s
    Response.Redirect("IntelligentQuery.aspx")
End Sub


Private Sub listbox2_SelectedIndexChanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles ListBox2.SelectedIndexChanged
    Dim st As String, itm As String
    Dim i As Int16, present As Int16 = 0
    st = ListBox2.SelectedItem.ToString
    For i = 0 To ListBox1.Items.Count - 1
        itm = ListBox1.Items(i).ToString
```

```
            If itm.equals(st) Then
                present = 1
            End If
        Next
        If present = 0 Then
            ListBox1.Items.Add(st)
        End If
    End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim i As Int16, j As Int16
        For i = 0 To ListBox2.Items.Count - 1
            Dim st As String, itm As String
            Dim present As Int16 = 0
            st = ListBox2.Items(i).ToString
            For j = 0 To ListBox1.Items.Count - 1
                itm = ListBox1.Items(j).ToString
                If itm.Equals(st) Then
                    present = 1
                End If
            Next
            If present = 0 Then
                ListBox1.Items.Add(st)
            End If
        Next
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        ListBox1.Items.Clear()
    End Sub
End Class
```

12.3 SCREEN SHOTS

Query

Students who are placed in CTS and members of rotaract club with percentage > 85

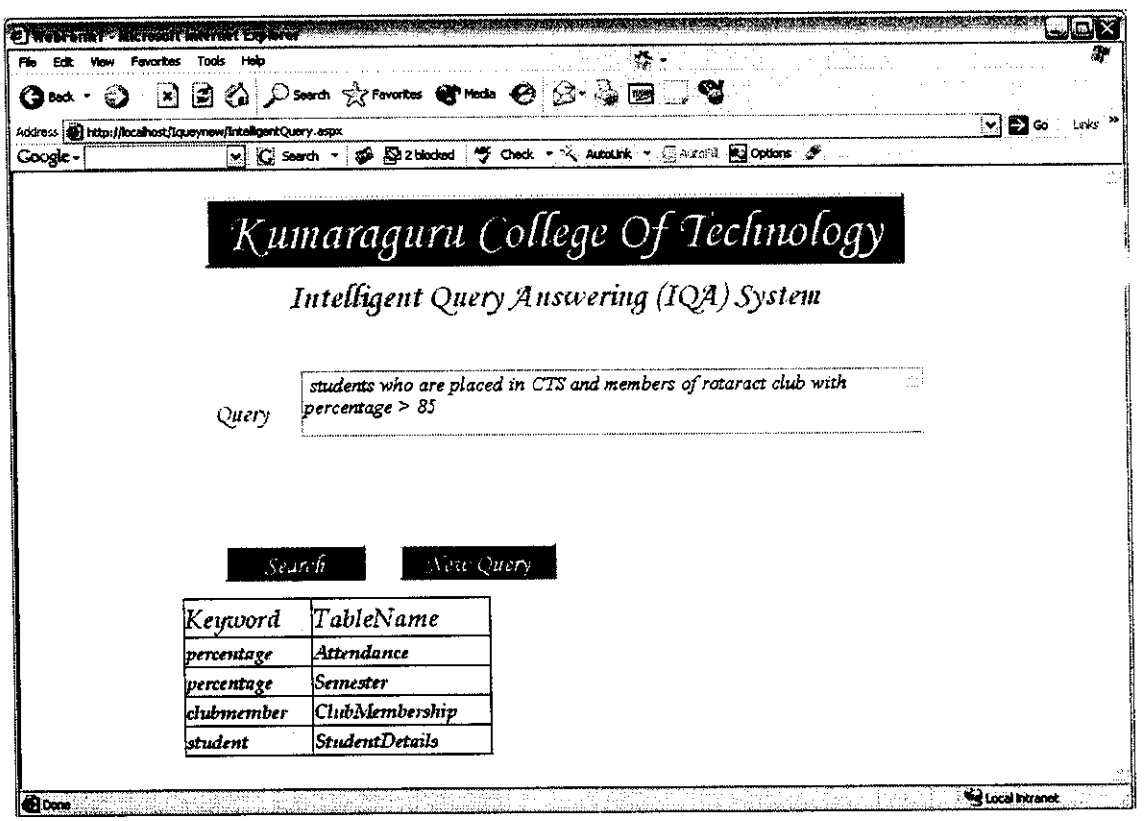


Fig. 12.1 tabdet

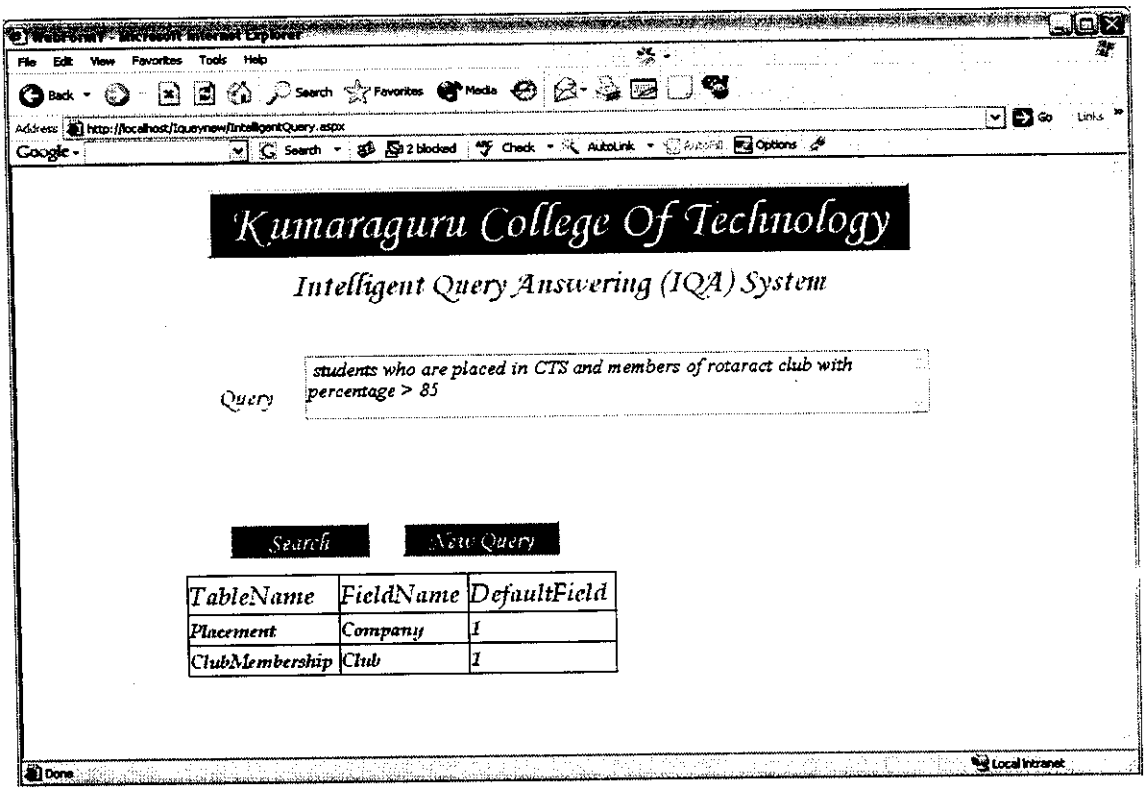


Fig. 12.2 fielddet

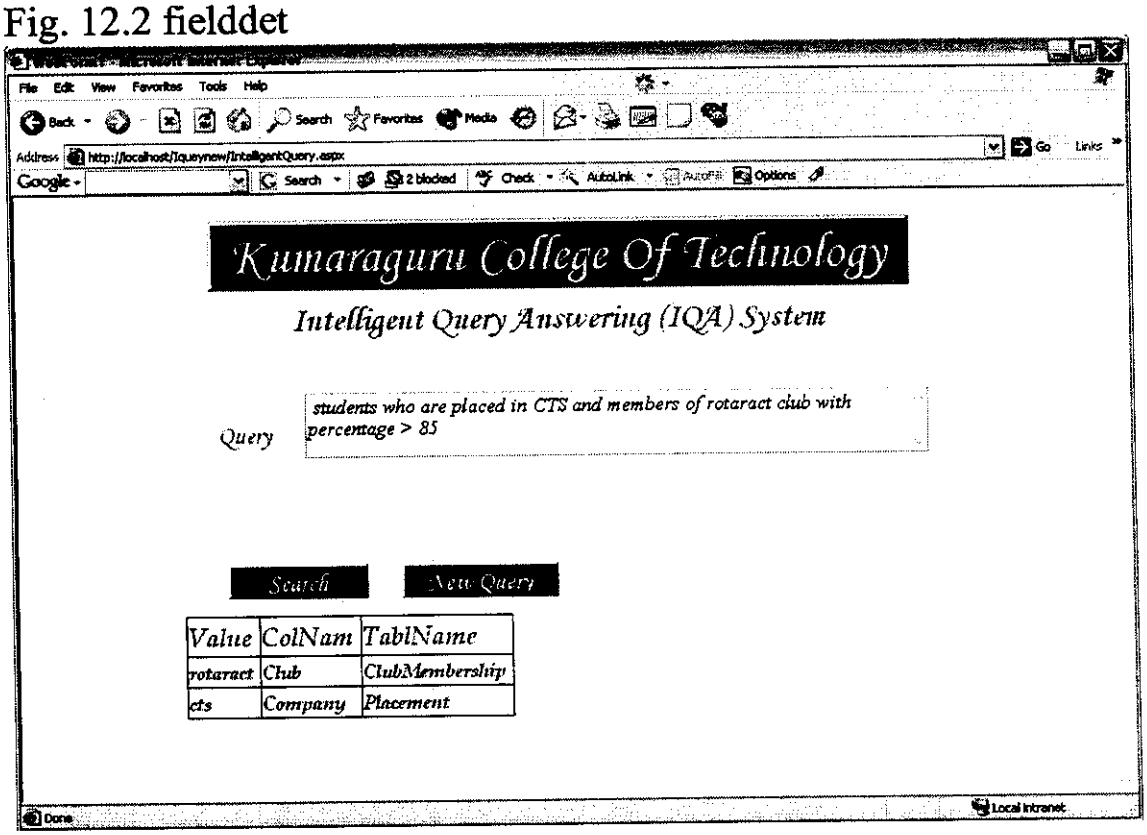


Fig. 12.3 hashvaltab

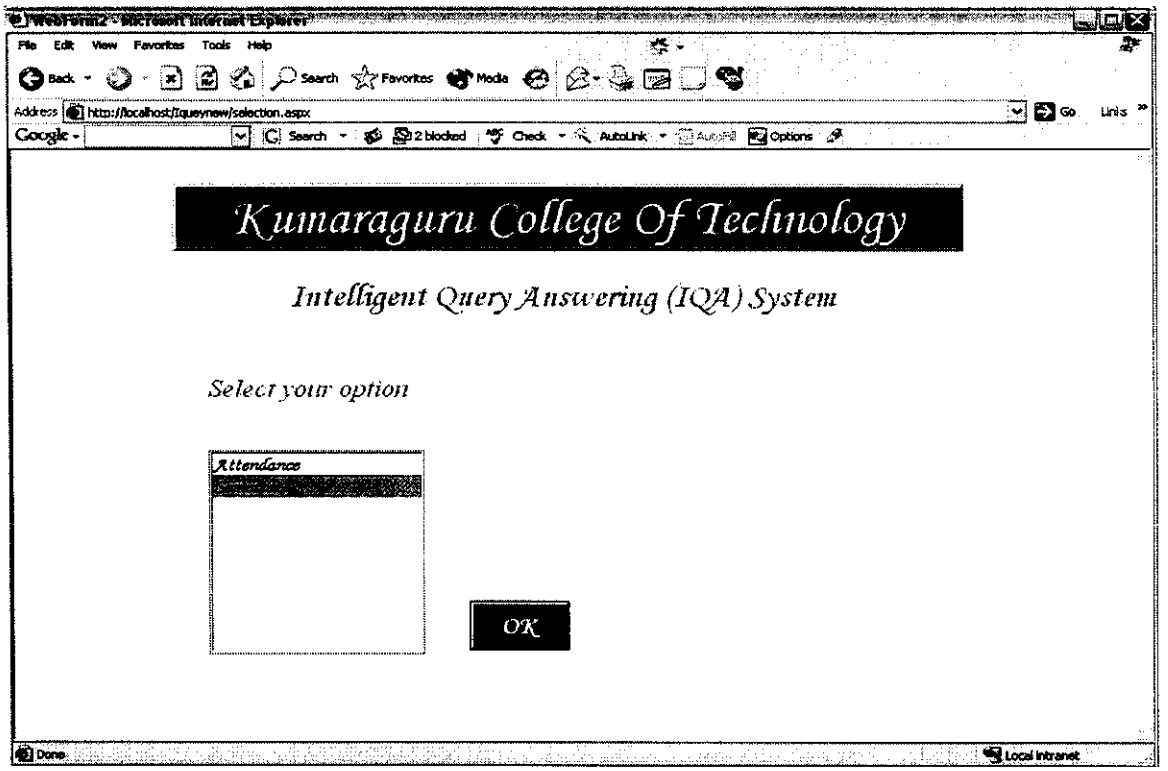


Fig.12.4 *Selection Page* to resolve keyword (percentage) ambiguity

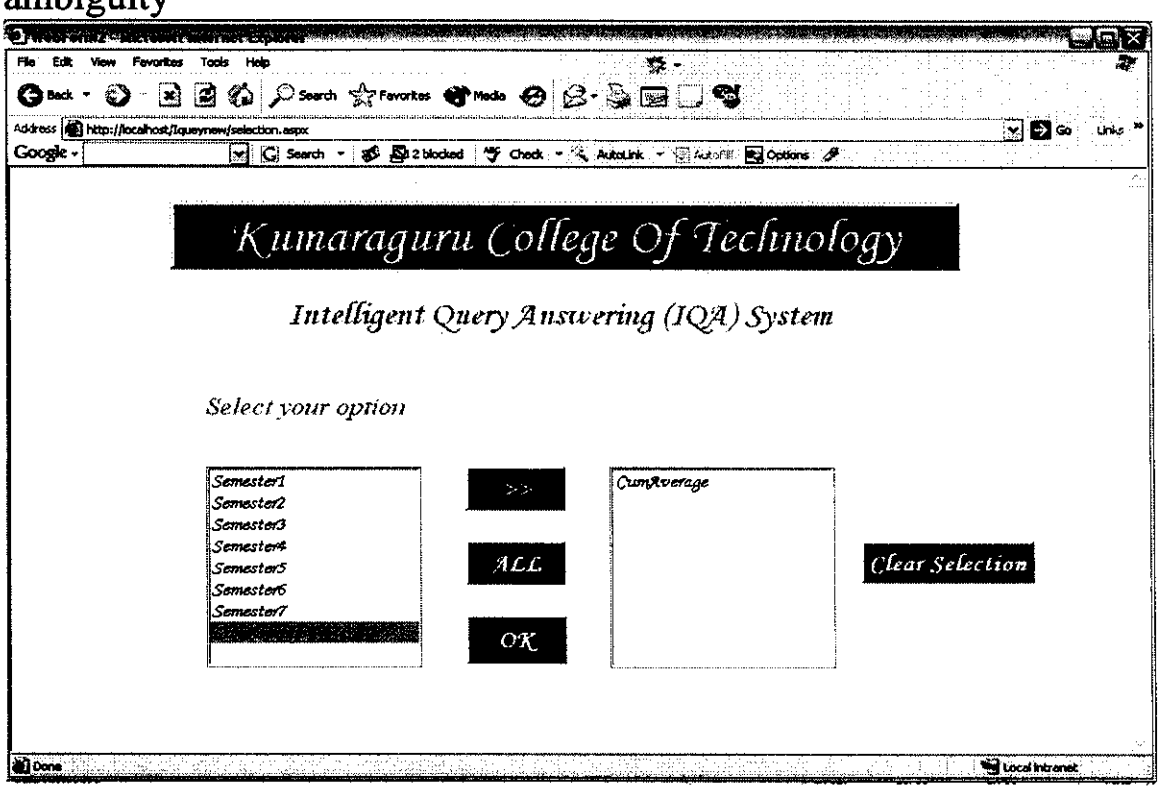


Fig.12.5 *Selection Page* to select the required field

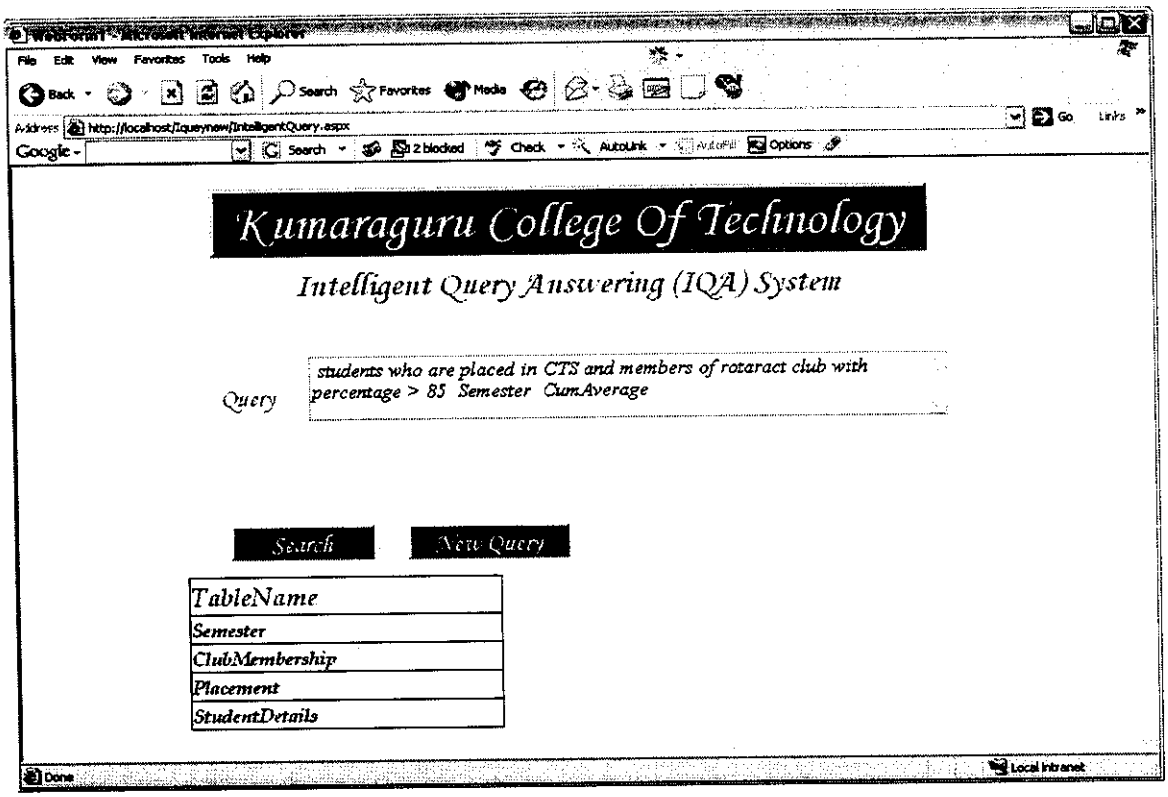


Fig. 12.6 finaltablst

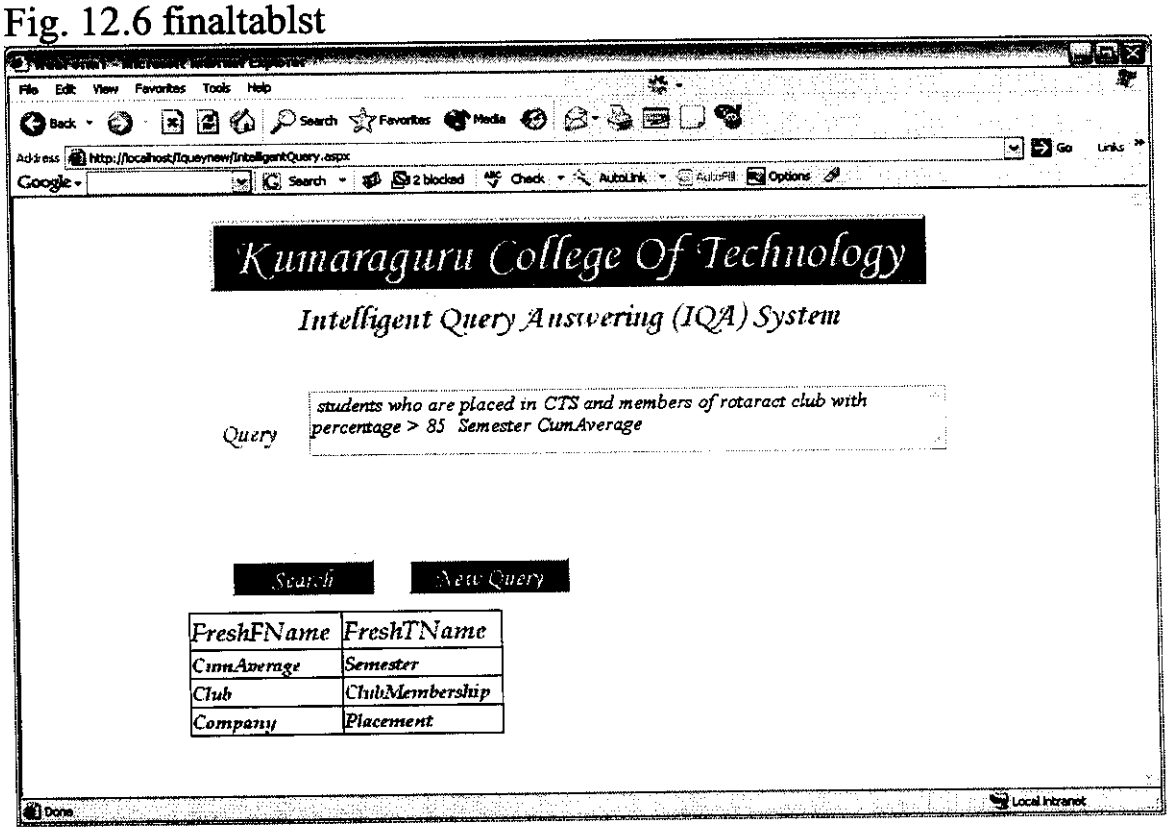


Fig. 12.7 freshfielddet

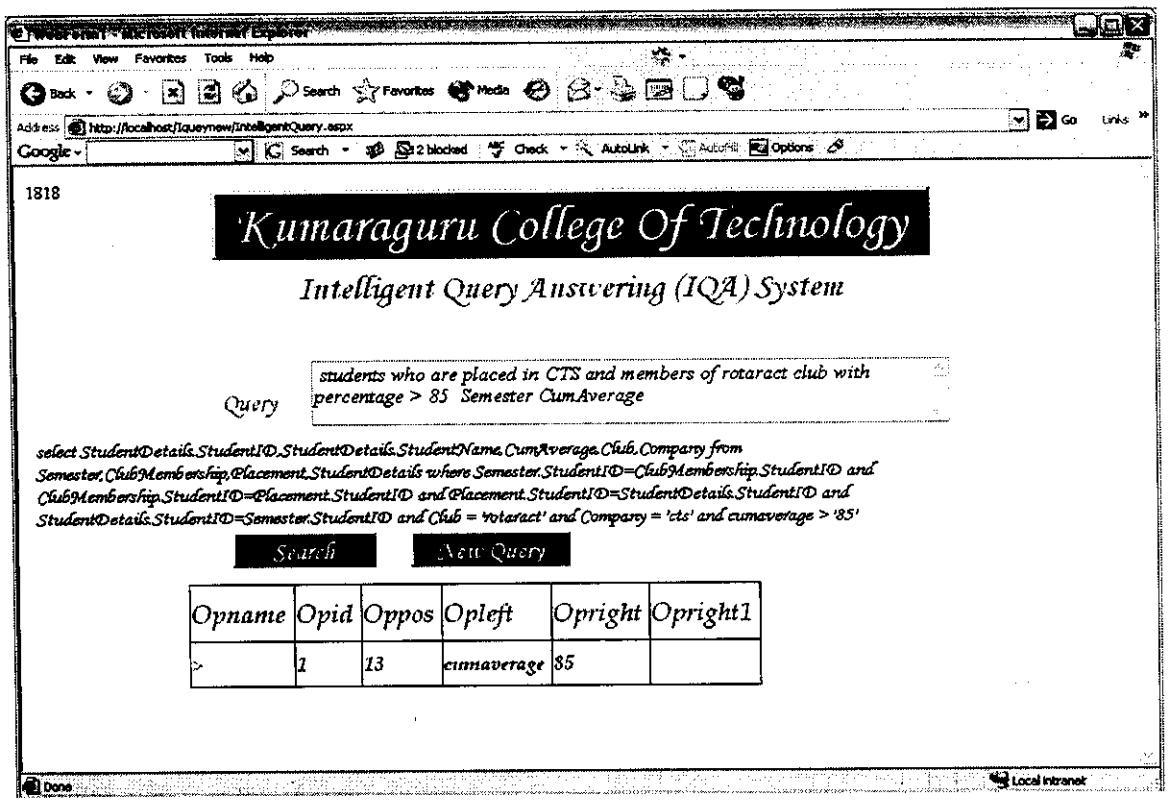


Fig. 12.8 combdet

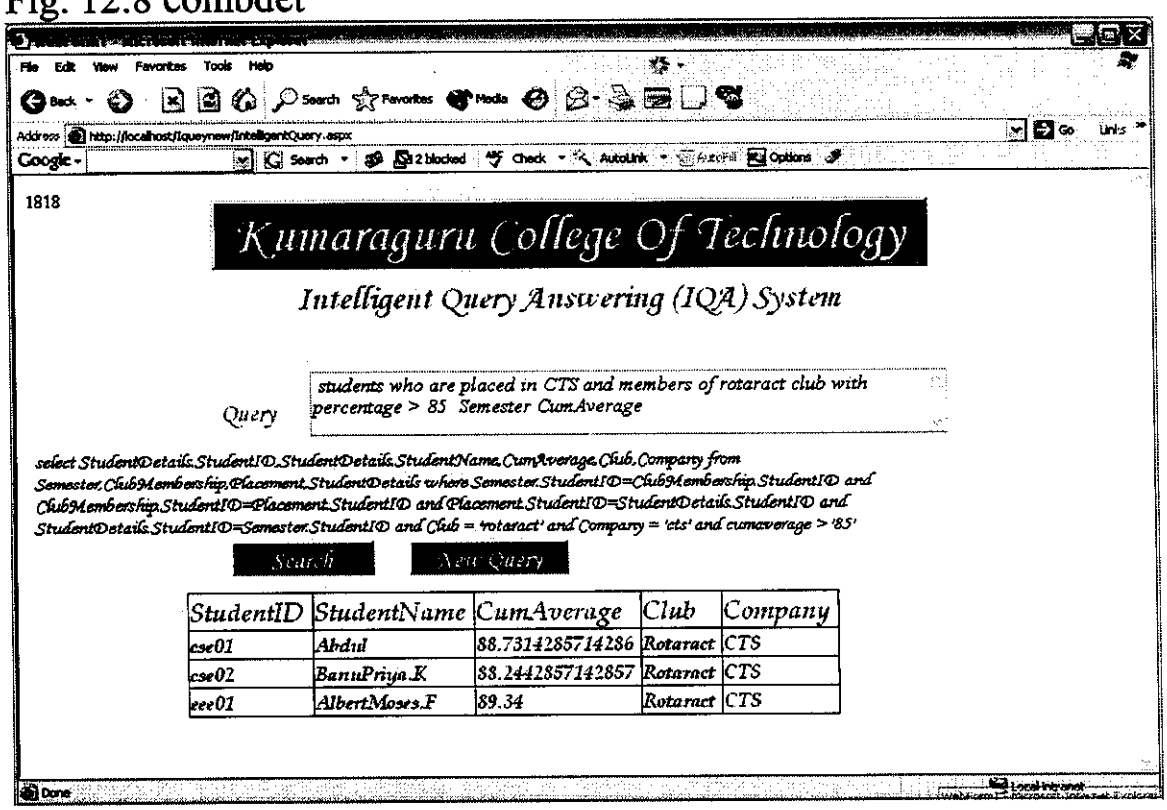


Fig. 12.9 Required set of tuples

Query

Abdul (only the student name)

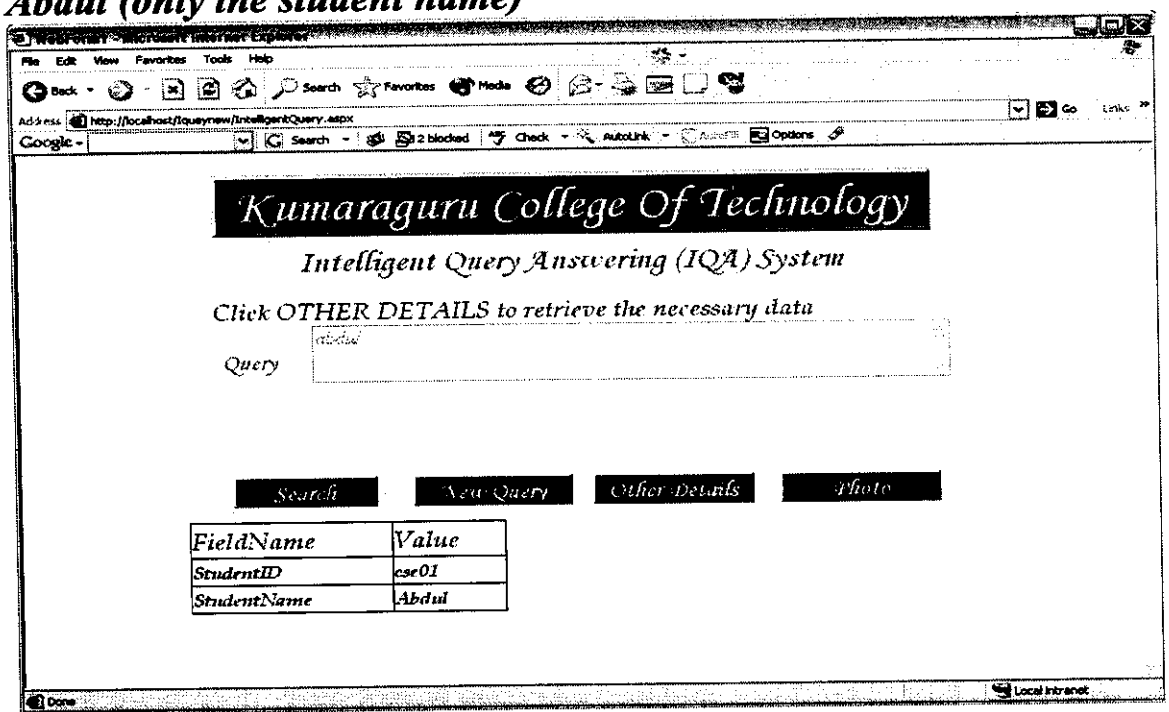


Fig. 12.10 Displaying name and ID of the student, the buttons *Other Details* and *Photo* are highlighted

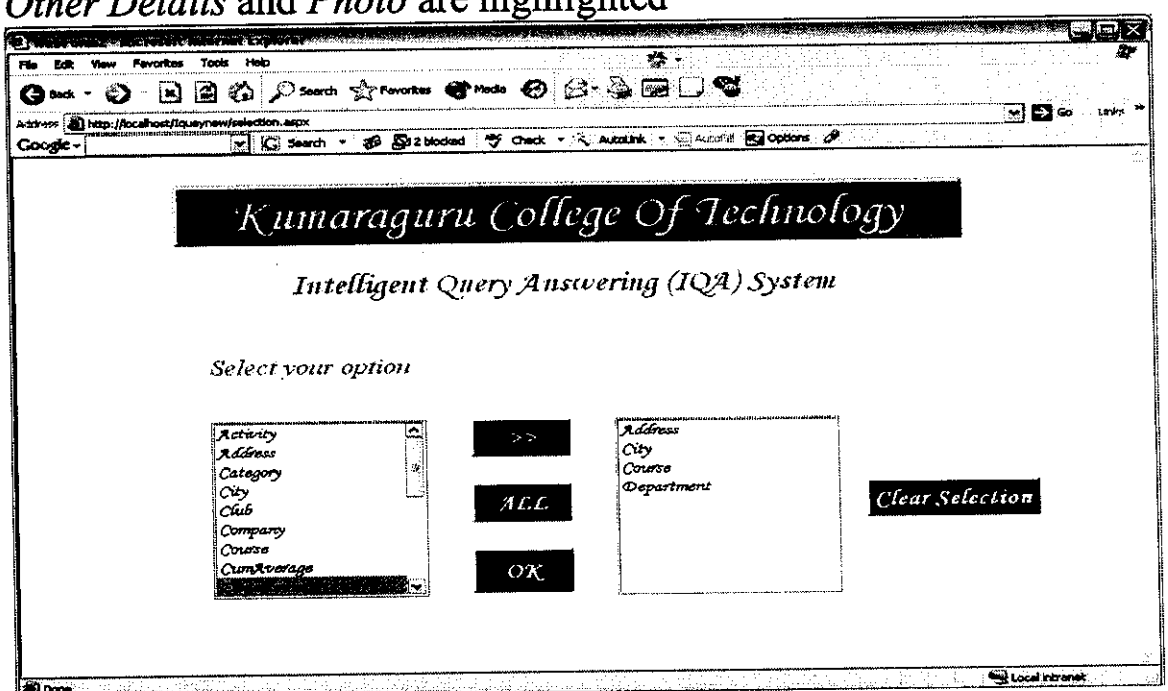


Fig. 12.11 Displaying *Selection Page* on clicking *Other Details* button

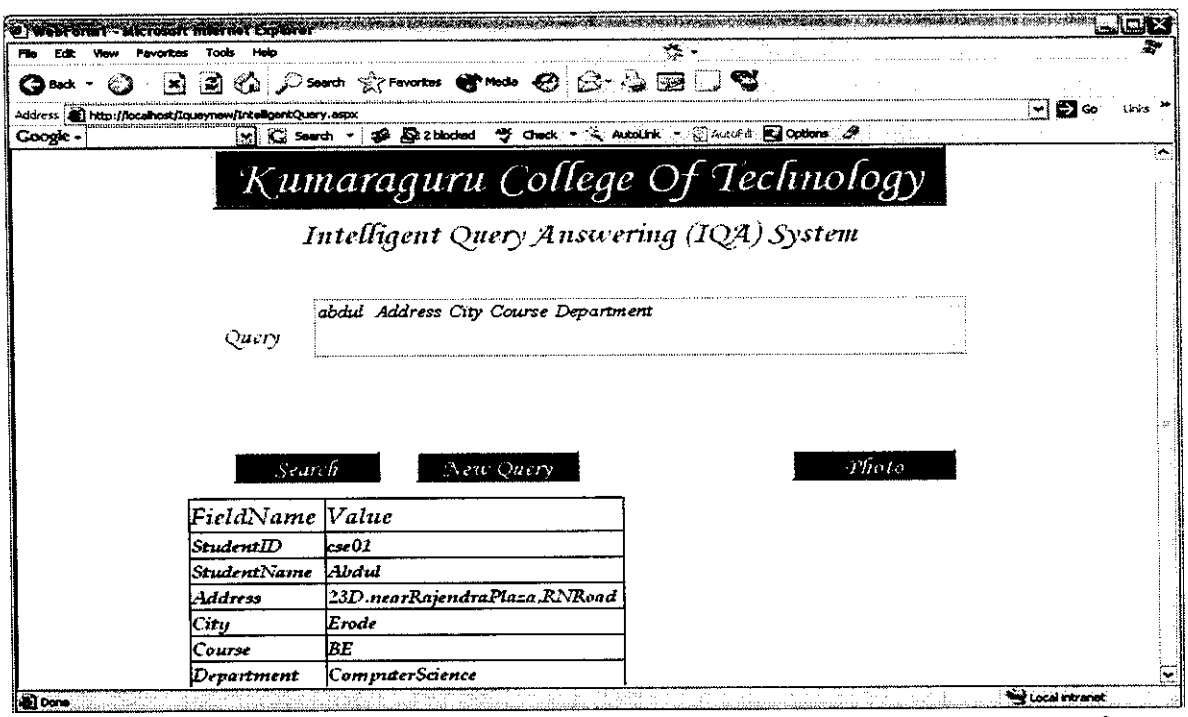


Fig. 12.12 Displaying the details selected in the *Selection Page* of the particular student

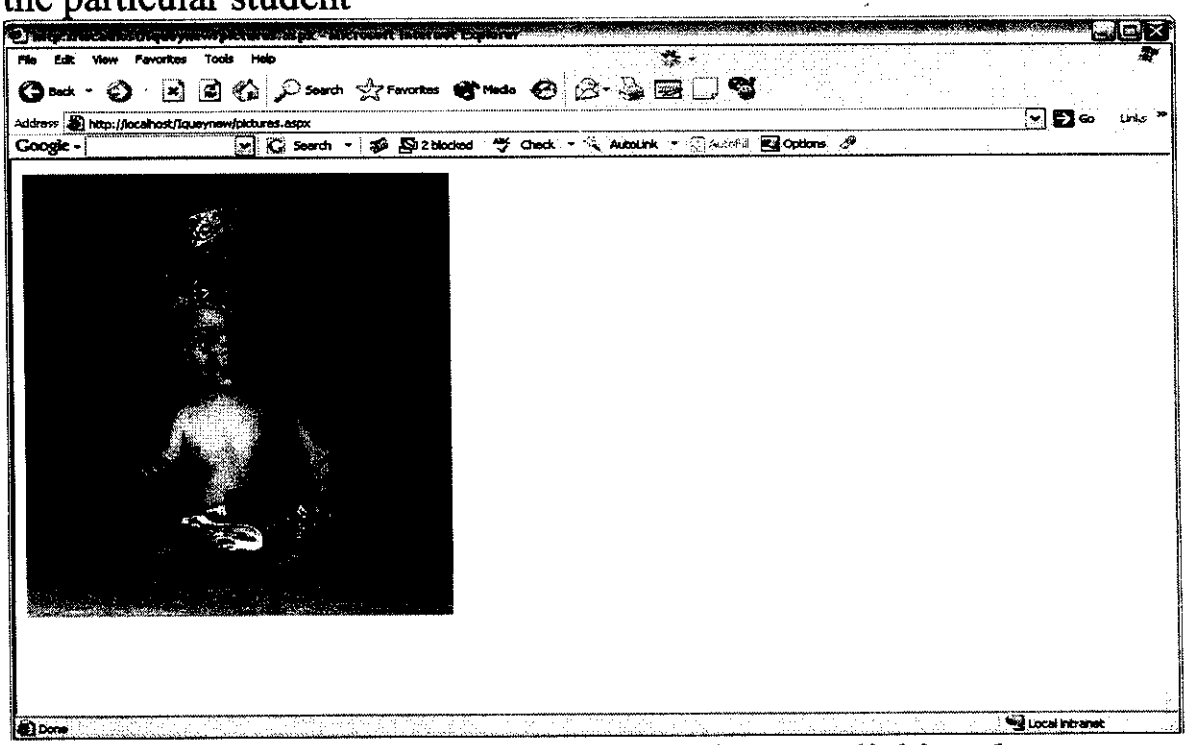


Fig. 12.13 Displaying the photo of the student on clicking the *Photo* button

REFERENCES

13. REFERENCE

13.1 PAPERS

[1] T.Y. Lin, Xiaohua Hu, Nick Cercone, Jianchao Han, 'Intelligent Query Answering Based on Neighborhood Systems and Data Mining Techniques' Proceedings of the International Database Engineering and Applications Symposium (IDEAS'04), IEEE 2004.

[2] Jiawei Han, Yue Huang, Nick Cercone, Yongjian Fu, 'Intelligent Query Answering By Knowledge Discovery Techniques', Work for The Natural Sciences and Engineering Research Council Of Canada and Center for Systems Sciences of Simon Fraser University.

[3] Sara Cohen, Werner Nutty, Alexander Serebrenik, 'Algorithms for Rewriting Aggregate Queries Using Views'

[4] Doina Caragea', Jie Bao, Jyotishman Pathak and Vasant Honavar, 'Ontology-based information integration using INDUS system'

13.2 WEB SITES

www.vbdotnetheaven.com

13.3 BOOKS

i) Danny Ryan and Tommy Ryan, ASP.NET, Published by Hungry Minds

ii) Mridula Parihar et al (2002), ASP.NET Bible, Published by Hungry Minds

iii) Evangelos Petroutsos; Asli Bilgi, Mastering Visual Basic.NET Database Programming, pp 228-263

iv) Rational Testing Products, Rational Software Corporation