A Fuzzy Database Query System with a Built-in Knowledge Base

By

Chang Yu

August 1995

Supervised By

Dr. K. S. Leung & Dr. M. H. Wong

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Computer Science

The Chinese University of Hong Hong

Acknowledgement

I would like to express my deepest gratitude and appreciation to my supervisors Dr. K.S. Leung and Dr. M.H. Wong, who have given me invaluable guidance, advice and help during my study and research.

I also wish to thank my examiners, Dr. Ada W.C. Fu and Dr. Chin Lu, whose constructive comments have greatly help me in carrying out this project.

Finally, I am also grateful to all the friends I met in the Department of Computer Science, CUHK. Special thanks Ms. Wong Wai-ting, Mr.Chan Chi-lok, Mr.Lau Sau-ming and Mr. Mak Kei-fu. Their encouragement, companions and help made my life in CUHK delightful.

Abstract

This thesis is concerned with techniques for database queries involving fuzzy conditions and concepts. A fuzzy query system is developed, which possesses the following main capabilities:

1. It has a built-in Knowledge Base, which is designed to deal with queries involving highly-aggregated, imprecisely and vaguely-defined concepts that cannot be directly processed without being further decomposed into more specific conditions. In the knowledge base, each concept is represented by a hierarchical concept tree. Each node of a tree may be a sub-concept, an operator, or a fuzzy or crispy condition. The use of concept trees to form a knowledge base to handle general concepts is one of the main contributions of this thesis.

2. It has a Fuzzy Processor, which is designed to handle the fuzzy conditions contained in a user's query and the output of the knowledge base. The fuzzy processor supports six operators, which are shown to be equivalent to some decision making rules in the theory of decision-making under uncertainties. In addition to providing the standard functions such as AND, OR, etc., the operators are particularly useful when a user desires to search for good solutions from the database in a situation where there are a number of future events which occur with uncertainty.

3. The system supports the use of external functions, by applying the technique of dynamic library. This provides a flexible and powerful tool for the appropriate definition of membership functions from the application level. A data manager is built in the system to evaluate the external functions, which together with the Fuzzy

Processor offers an effective way to handle the fuzzy conditions and evaluate their relevant degrees of membership. The capability of the system to accept external functions is particularly useful when multidimensional membership functions have to be defined, which overcomes a difficulty in some previous approaches.

In addition, other modules such as a Parser, an Interfaces, etc., are also developed. The design and the implementation of the overall system consisting of these modules are described in details in this thesis. The system is tested and evaluated on an experimental database, which suggest that the system has the expected functions, which can accept not only well-defined fuzzy conditions as well as ordinary crispy conditions, but also highly-aggregated, vaguely-defined concepts.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Motivation and Objectives

In traditional database management systems, queries for retrieving data are usually defined by using crispy conditions. For example, in the Structured Query Language (SQL), data retrieval operations are defined in the following form:

Select attribute-list from relation where predicate,

where *attribute-list* specifies attribute values to be returned to the user, *relation* identifies tables in the relational database, and *predicate* specifies the conditions of the query consisting of Boolean expressions. The requirement that conditions be defined by Boolean expressions implies that for every candidate stored in the database, a binary decision should be made by the querying system regarding whether the candidate is exactly an answer to the user or not. Clearly, Boolean query logic may represent only crispy conditions, like `age < 30`, `salary > HK$40,000`, etc.

In many practical situations, however, database queries unavoidably involve imprecise, fuzzy conditions or concepts. Some typical examples are shown below:

1. In business decision making, it is often desirable to search for some good solutions under certain fuzzy, imprecise criteria. For instance, a query to a database in stock market may be submitted with the following criteria: find those stocks in

the electronic industry that performed excellently in the past year. In this case, the query involves crispy conditions `year = 1994` and `industry = electronics` as well as a very general, vague concept of `excellent performance`.

2. In engineering design problem, it is often desirable to start the design of a new part with a similar part that has already been developed, instead of developing the new part from scratch. In such a case, it will be necessary to search for those similar parts that are stored in an engineering database. The retrieval of candidate parts from the database will be subject to a fuzzy concept that they are `similar` to the new part under development.

3. To recruit employees to fill in certain positions, a query to retrieve candidates from a database maintained by a job center may include criteria like: female, young, with an advanced degree in computer science or a related discipline, experienced, can speak Japanese, etc. In this case, it is clear that the conditions `female`, and `can speak Japanese` are crispy, whereas `young`, `a computer-related degree`, and `experienced` are fuzzy conditions. On the other hand, someone looking for a job may submit a query with the following conditions: a job as a system analyst, high pay, good benefits, working place near his home, etc. In this case, the crispy condition is `system analyst`, whereas the fuzzy conditions are `high pay`, `good benefits`, and `near home`.

It is easy to find that various practical problems exist in engineering, business, academy, government, etc, which are of the same nature as above. To respond to the requirements of data retrievals in such situations, it is clear that more sophisticated querying systems with the capability of processing imprecise, fuzzy queries are needed. This constitutes the main motivation of our research in this thesis.

In recent years, lots of efforts have been made, aiming to propose some effective approaches and systems to tackle the fuzzy database query problem. We will review, in the next Chapter, some of the works which are relevant to our research. In general, as fuzzy

database query is still a new area of research which has not become mature yet, many technical problems still remain to be resolved, and each approach or system proposed so far still has its limitations and disadvantages although at the same time they have offered certain promises to solve some aspects of the problems. The main objectives of our research are to tackle some of these important problems. Technically, our main objectives are as follows:

1. A main problem found to exist in almost all systems that have been developed for handling fuzziness in database queries is that they require users to completely specify the explicit definitions of fuzzy conditions and their membership functions. Nevertheless, in many cases a query submitted by a user may involve certain highly-aggregated, imprecisely and vaguely-defined concepts, as we have discussed above. These concepts cannot be directly processed without being further decomposed into more specific and well-defined conditions. A query system usually does not possess such a capability even if it can handle fuzzy conditions, since the further decomposition and definition of general concepts often require some high-level knowledge on the concepts from experts. An integrated system that has the capability of mapping general concepts into well-defined fuzzy/crispy conditions by using expert knowledge is mostly desirable. To develop such a system will be one of our main objectives.

2. Another main objective of our work is to design and develop a query system which can be added on to an existing database management system like ORACLE, with minimum disturbance to the existing data management environments. In other words, we assume that the database and its management are conventional (i.e. nonfuzzy) and our querying system is used to make the data retrievals be able to accommodate fuzzy concepts/conditions without any further change and modification on the existing environments. Note that such an 'add-ons' feature is important in making a query system more acceptable and applicable, which is one of the basic lines of research in fuzzy database query, see Kacprzyk, Zadrozny, and Ziolkowski [20].

3. To apply fuzzy set theory and decision making theory to process more effectively fuzzy conditions is the third objective of this research. The fuzzy conditions will be expressed in terms of membership functions. To provide more flexibility to express the membership functions, our system will be designed to have the capability of using external functions for the representation of membership functions. This will also give the users more flexibility to perform other desirable operations.

1.2 Outline of the Work of This Thesis

The main work of this thesis is to develop a database query system to process database queries involving fuzzy/imprecise conditions and/or concepts. Specifically, our main work consists of the following:

1. We propose to use a knowledge base to handle general fuzzy concepts. The knowledge base is a collection of concept trees. Each concept tree represents a concept, in which the most general concept is placed at the highest level node (the root) of the tree, whereas specific (crispy or fuzzy) conditions and subconcepts (each of which may also be defined by another concept tree) are located at the lower level nodes. This provides an effective approach to decompose a general concept into more specific conditions in accordance with the knowledge on the concept.

2. We develop a fuzzy query language to process fuzzy conditions, by applying the fuzzy sets theory and the decision-making theory. Six operators are used, which play different roles according to the need of the users. Although some operators were presented and used in previous literature, we derive the operators from a different perspective, i.e. decision-making under uncertainties, which will better represent the user's need in data retrievals under similar nature of criteria. A Parser and a Fuzzy Processor are incorporated in the query system, which first transform fuzzy conditions into some new crispy conditions, and then use the fuzzy operators to

process the candidates retrieved from the database under all the crispy conditions. In this process, the standard SQL language is used and all the operations of the database management system (ORACLE) remain unchanged.

3. We propose to use external functions to represent membership functions and to perform other desirable operations. For this purpose, a Data Manager is built up in the query system to provide this capability. The provision of the use of external functions allows the system to be able to perform various tasks easily and effectively. In particular, it provides a very flexible way to represent the membership functions of fuzzy conditions as compared to previous systems.

The query system with the above capabilities has been designed, developed, and tested on some sample databases. The overall architecture and design of the system as well as the development and implementation of its various modules will be described in details in this thesis. Meanwhile, our approaches to solve various problems in developing such a system will be presented.

1.3 Organization of the Thesis

The organization of the thesis is as follows: Chapter 2 below will be a review of the previous relevant work. In Chapter 3, we will first introduce some necessary concepts on the fuzzy set theory, and then develop our fuzzy database query language, including its syntax and its fuzzy operators to process fuzzy conditions. In Chapter 4, the design and the functions of the system and its modules will be described in details. Chapter 5 will discuss some key issues in the implementation of the system, in particular the modules of the Knowledge Base and the Data Manager. To evaluate the performance of the system, in Chapter 6, we will consider some problems in job application/recruitment. Several cases will be used to show the capabilities of the system. Chapter 7 is conclusion, in which we will summerize our work and provide some comments on future research.

Chapter 2

REVIEW OF RELATED WORKS

This Chapter reviews some representative early works in the field of imprecise/fuzzy database query, from which we see the various difficulties existing in the imprecise/fuzzy database query problem, and the efforts that researchers in this field have expended aiming to overcome the difficulties. Our work in this thesis is in fact a continuation of some of these efforts and a further development of some of the approaches proposed.

2.1 Deduce2

Chang [12] developed a system, called Deduce2, which aims at an extension of Deduce, a deductive system which provides users with a predicate calculus language. Deduce2 accepts a query involving two parts connected by an AND: a Boolean condition C1 and an optional imprecise condition C2. The imprecise condition C2 may refer to terms like *old*, *young*, *good*, etc., which are combined using the two usual operators: AND and OR. The semantics of a query in the system may be stated as "rank according to C2 the tuples which satisfy C1".

The core of Deduce2 is its ranking mechanism. It assumes that any imprecise term involved is a monotonic function of one attribute, which may be a base attribute (present in a relation), or a derived attribute (computed from attributes of relations). Thus, a term '*old*' can be represented as an increasing function of the base attribute *age*, and a term

'*GPA around 3.0*' can be represented by a decreasing function of the derived attribute $|GPA - 3.0|$. The adequacy under an imprecise term is then measured by the rank obtained by sorting the tuples (having been retrieved by using C1), which is performed increasingly or decreasingly depending on whether the imprecise term is a decreasing or increasing function of the attribute, with the convention that *the smaller the rank, the better the satisfaction of the imprecise term.* If C2 consists of two terms T1 and T2 (assume that they are monotonic functions of attributes A1 and A2 respectively), then, for a conjunction T1 AND T2, a tuple will be first assigned with a rank R1 under T1 and a rank R2 under T2, and then the final rank is max(R1,R2). Converserly, for a disjunction T1 OR T2, the final rank will be min(R1,R2).

For example, consider a query "select the names of those Ph.D students in computer science department who are old and whose GPA is around 3.0". The imprecise part of the query is: T1 = *old* (increasing function of age) AND T2 = *GPA around 3.0* (decreasing function of $|GPA - 3.0|$). Suppose the following three candidates are retrieved from the database under the precise condition 'Ph.D students in computer science': (Lee, 32, 2.8), (Ng, 38, 3.6), (Wong, 35, 2.2). Clearly the rank under T1 is: Ng, Wong, Lee, whereas the rank under T2 is: Lee, Ng, Wong, which gives Lee(3,1), Ng(1,2), and Wong(2,3). Thus, the final rank is that Ng is the best, Lee and Wong are the second. If the operator AND is changed to OR, then the final rank is that Lee and Ng are the best, Wong is the second.

Remark. This system represents an early work aiming to handle imprecise queries. It is basically founded on the usual Boolean query logic and its objective is clearly to extend an existing database management system so that it can have the capability to process imprecise queries. Nevertheless, the queries allowed by the system are not general since

1. the introduction of the imprecise part only aims at the rank of the tuples retrieved under the crispy conditions;

2. any imprecise term must be supported by a monotonic function.

Besides, it should be noted that the expression of imprecision does not take enough

semantic aspects into account. In the example above, when the ages of the candidates Lee, Ng, and Wong were 2, 8, and 5, respectively, the final result of rank would have remained the same. This is a surprising result as the age condition is not satisfied by any of the three candidates at all.

2.2 ARES

ARES is a system developed by Ichikawa and Hirakawa [17], which uses a *similarity* operator denoted as $\approx$ to represent an imprecise predicate 'more or less equal to'. To measure the satisfaction of an imprecise condition $A \approx a$, a distance $d(v_1, v_2)$ is defined and the value of $d(A, a)$ is evaluated. Then, A will be regarded as being very similar to a if the value of the distance is close to zero. Given an acceptance threshold t, the system will retrieve those tuples from the database whose values of distance are less than t.

If a query consists of Boolean and imprecise predicates connected by the operator AND, the system adopts the following process to handle the query:

1. The user is asked to supply a threshold t_i for each imprecise predicate C_i.

2. Based on the definition of distance, the system converts each imprecise condition C_i into some Boolean conditions, under the convention that any tuple retrieved from the database will have a distance less than the threshold t_i.

3. A new query consisting of all the old and new Boolean conditions will be used to select acceptable tuples for which each elementary distance is less than t_i and the global distance is less than an acceptable value, where the global distance is defined as the sum of the elementary distances of the imprecise predicates C_i for all i.

4. The system sorts the tuples in descending order of the global similarity, and then returns them to the user, with their corresponding global distance as the indicator of overall satisfaction.

Consider an example, where the query is to select those students whose ages are around 20 and whose GPAs are about 3.0. Assume that the distance of age from an expected value a is defined as:

$$d(age, a) = \begin{cases} 0, & \text{if } |age - a| = 0, \\ 1, & \text{if } 0 < |age - a| \leq 3, \\ 3, & \text{otherwise.} \end{cases} \tag{2.1}$$

And the distance of GPA from an expected point b is defined as:

$$d(GPA, b) = \begin{cases} 0, & \text{if } |GPA - b| \leq 0.1, \\ 1, & \text{if } 0.1 < |GPA - b| \leq 0.2, \\ 2, & \text{otherwise.} \end{cases} \tag{2.2}$$

Suppose that the following data are stored in the database:

(Chan, 21, 3.0),
(Li, 17, 2.7),
(Siaw, 24, 2.9),
(Tang, 20, 2.7),
(Wong, 23, 3.1),
(Yiu, 20, 2.9).

Note that $a = 20$ and $b = 3.0$ in this example. Thus, using the two forumula (2.1) and (2.2) above, we can compute the distances of these candidates under the two conditions regarding age and GPA, which are shown in the Table 2.1 :

Assume that the threshold on age supplied by the user is 2, the threshold on GPA is 1, and the global threshold is 1. Then, it is cleat that Chan, Li, Tang, Wong, and Yiu satisfy the condition on age, while Chan, Siaw, Wong, and Yiu satisfy the condition on GPA. But overall, only Chan, Wong and Yiu satisfy the restriction on the global distance. Applying ARES, the following results will be obtained:

Chan[1], Wong[1], and Yiu[0],

where the number inside the bracket indicates the global distance. Therefore, Yiu will be

Table 2.1: Distance

Name	$d(age, 20)$	$d(GPA, 3.0)$
Chan	1	0
Li	1	2
Siaw	3	0
Tang	0	2
Wong	1	0
Yiu	0	0

ranked as the best, and Chan and Wong the second.

Remark. This system also aims to handle the problem of imprecise query. However, the semantics of a query is not straightforward in this system. In fact, the results generated by the system strongly depend on the threshold values, and a pair of candidates will be regarded as the same if they are all within the threshold values. For example, Chan and Wong are of the same rank even though it is obvious that Chan satisfies the conditions better than Wong does. In addition, Tang is not selected even if he is very close to the best candidate Yiu.

Another problem with ARES is that the meaning of the global similarity is not very obvious. Besides, it is sensible only when all the predicates are connected by ANDs.

2.3 VAGUE

VAGUE is a system developed by Motro [31], which can be regarded as an improvement of ARES. In dealing with a *similarity* condition $A \approx a$, a distance is also defined. Specifically, a distance between v_1 and v_2 for each predicate C_i is defined subject to the following constraints:

$$\begin{cases} d_i(v_1, v_2) \geq 0, & \forall v_1, v_2, \\ d_i(v_1, v_2) = 0, & \text{if } v_1 = v_2, \\ d_i(v_1, v_2) = d_i(v_2, v_1), & \forall v_1, v_2, \text{ and} \\ d_i(v_1, v_2) \leq d_i(v_1, v_3) + d_i(v_3, v_2), & \forall v_1, v_2, v_3. \end{cases}$$

Then, a radius r_i is supplied by the user to each similarity condition $A \approx a_i$, which is considered being satisfied if $d_i(A, a_i) \leq r_i$.

However, unlike ARES, VAGUE adjusts the raw distance under each predicate C_i when considering the combination of the predicates. Specifically, it introduces an *adjusted distance* as follows:

$$\hat{d}_i(A) = w_i \frac{d_i(A, a_i)}{r_i},$$

where w_i is the weight associated with the predicate C_i. (Note that the *adjusted distance* applies to a Boolean condition C_i by default. In this case, $\hat{d}_i(A) = 0$ if A satisfies the condition and $\hat{d}_i(A) = \infty$ otherwise.)

In addition, VAGUE allows AND as well as OR. For a condition $OR(C_i, C_j)$, the adjusted distance for a tuple is the smallest of the adjusted distances related to C_i and C_j. Finally, the global distance of the combination of conjunctions $AND(C_1, ..., C_m)$ is obtained as the root of the sum of the squares of the adjusted distances corresponding to each C_k, $k = 1, 2, ..., m$. The global distances for all the tuples satisfying the Boolean conditions will be evaluated and those tuples whose global distances are less than an overall threshold value will be regarded as qualified answers to the user's query, and are returned to the user with a rank made in accordance with their global distances.

Remark. VAGUE is an improvement of ARES as it performs a normalization on the distances by introducing the *adjusted distances*. This makes the final results not so sensitive to the theroshold values as compared to ARES. Besides, it allows the use of the operator OR. Nevertheless, like ARES, VAGUE regards all candidates as the same when they

all fall in the limits governed by the threshold values (the radii). Moreover, the system only supports imprecise query involving the concept of *similarity*. It does not have the capability to deal with other imprecise conditions like *young*, *much more than*, *excellent*, etc.

2.4 Fuzzy Sets-Based Approaches

It can be seen that the systems as reviewed above do not take enough semantic aspects into account, which care only about certain extreme points in a query. This often gives some surprising results like two very different candidates are regarded as the same, or a candidate is rejected even if it is very close to the best qualified candidate. To resolve these problems, since the late 1970s, a new direction of research has been created based on the theory of fuzzy sets (Zadeh[43]). Tahani [37] is among the first people who advocate the use of fuzzy sets theory for imprecise database query. Tahani suggests the extension of the relational query language SEQUEL to support fuzzy queries. In his original proposal, a fuzzy relation is defined by associating a degree of membership μ with each tuple, to represent its satisfaction to the fuzzy predicates that are allowed to be expressed in a query as linguistic variables such as *old*, *high*, *small*, etc. These predicates can be connected by operators AND and OR working as MIN and MAX. The final results returned to a user will be those tuples with a non-null degree of membership in terms of satisfying the combination of all the predicates. Tahani's approach can be regarded as one of the significant attempts in the early time to apply fuzzy sets to attack the imprecise query problem.

In order to extend the imprecise query capability of a relational DBMS, Kacprzyk and Ziolkowski [21] and Kacprzyk, Zadrozny and Ziolkowski [20] attempt to define the meaning of a set of queries involving the so-called quantified queries. An example is the query 'find the names of those students such that *almost all* the conditions {young, high GPA, live near the campus, ...} are satisfied'. In general, they aim to attack the problem of imprecise

query in the form 'find those tuples such that P among the conditions $\{C_1, C_2, ..., C_n\}$ are as desired', where a condition C_i may be a precise one or an imprecise one, and P is either an absolute quantifier, such as *a dozen, nearly 100, etc.*, or a relative quantifier, such as *almost all, a few, most, etc.*, which are represented by using fuzzy sets.

Wong and Leung[42] propose a fuzzy query language, which is designed to retrieve data from a DBMS – VAX Rdb/VMS. A fuzzy information retrirval module is developed, which functions as an interface between users and the database system so as to support the fuzzy query language for users to retrieval data under both Boolean conditions and fuzzy predicates. The system works as follows:

1. Initially, a user's query is translated into the query language of the Rdb/VMS database system.

2. The Callable RDO facility of the Rdb/VMS is invoked to retrieve the required data from the database.

3. The data retrieved are evaluated against each fuzzy condition to obtain the degrees of membership in terms of satisfying the corresponding fuzzy condition.

4. Four operators, namely, AND, OR, COMB, POLL (see next Chapter for the details of these operators), are applied to derive the overall degree of membership of each record in terms of satisfying the combination of the fuzzy conditions in the user's query.

5. The records with non-null overall degree of membership will be ranked and returned to the user.

This system relates the fuzzy database query to a multi-criteria decision-making problem. Thus, it not only serves as an effective language for fuzzy database query, but also provides a powerful tool to assist decision-makers to find those optimal (or good) solutions from the database.

Datacycle is a database processing system that uses filtering technology to perform an efficient, exhausive search of an entire database. Recently, Mansfield and Fleischman [29] have described an approach which allows Datacycle to include fuzzy predicates in its query processing. They are mainly concerned with the problem of how to ensure the transaction processing efficiency in a high-volume query processing environment. For this purpose, they propose several techniques aiming to reduce the time requirement in evaluating the membership functions. One of them is to create a library of commonly used membership functions parameterized by the trapezoid breakpoints (the technique of using trapezoid points as a representation of membership functions is also investigated in Kacprzyk, Zadrozny and Ziolkowski [20]). Then during query parsing, breakpoints are substituted for the linguistic membership functions contained in the query. Besides, to support the use of ad hoc definition of membership functions from the application level, they allow the breakpoints to be specified within the grammer of their extended SQL. Nevertheless, their system limits the membership functions to piecewise linear functions only.

In addition to the works reviewed above, numerous approaches, mostly on the theoretical aspects of fuzzy query and its related issues, have been proposed in the literature. These include Umano [41], Zemankova and Kandel (1985), Prade and Tesemale (1987), Buckles and Perty [11], Bosc, Galibourg and Hamon [4], Kamel and Hadfield [22], Takahashi [38], Bosc and Pivert [7], etc.

2.5 Some General Remarks

1. In general, as we mentioned in Chapter 1, fuzzy/imprecise database query is still a new and active area of research, and many related issues still remain unresolved. Those traditional approaches like Deduce2, ARES, and VAGUE attempt to extend the capabilities of existing DBMSs so as to handle queries involving certain imprecise

predicates. However, each of them is more or less subject to some kinds of technical restrictions, as we have discussed above. In particular, none of these approaches takes enough semantic information into account, and the results generated by them are therefore highly sensitive to certain threshold points subjectively provided by the users. In some cases, a slight change of some threshold points may greatly change the final results.

2. The theory of fuzzy sets offers an ideal approach to overcome the difficulties encountered by the traditional approaches, as the use of fuzzy sets provides a systematical and accurate way to express imprecision and vagueness. Besides, many standard results that have already been derived within the general framework of fuzzy sets theory provides many useful tools for the development of query systems based on the concept of fuzzy sets. All these serve as the main reason why the notion of applying the fuzzy sets theory to the field of fuzzy/imprecise database query has become an exciting and active line of research in recent years. Nevertheless, although many interesting results have been obtained, numerous problems still exist.

3. A problem which exists in almost all previous approaches is that they presume that any fuzzy predicate presented in a user's query has been described at such a specific level that it can be directly represented by a certain fuzzy set, and the membership function of the fuzzy set can be directly supplied by the user or can be directly found in the system. This is actually not the case in many practical situations. As we have seen from the examples discussed in Chapter 1, in many situations the queries of users often contain some highly-aggregated, imprecise and vaguely-defined concepts, which cannot be processed without being further decomposed into more specific conditions. How to process these kinds of queries is a challenging problem. Considering that the decomposition of a highly-aggregated general concept usually requires the knowledge of relevant experts, we believe that the use of knowledge base could be a promising approach to tackle the problem. In this thesis, we will report our results in building up such a system with this idea.

4. Some proposed approaches aim to enhance some existing query languages like SQL to support directly imprecise queries (see, for example, Bosc, Galibourg and Hamon [4], and Bosc and Pivert [7]), but mechanisms to process imprecise predicates are absent in current relational databases and are still the subject of current database research, as pointed out by Mansfield and Fleischman [29]. There are two basic lines of research, according to Kacprzyk, Zadrozny and Ziolkowski [20], in the use of fuzzy sets in database query. One approach is to build databases and their management systems so that they can involve imprecision represented by fuzzy sets. In such a system, querying, updating, etc., can be carried out based directly on fuzzy sets. The other approach is to assume that the databases and their management systems are conventional, and aim to build some fuzzy processing systems on top of the existing systems to make them capable of processing imprecise queries. Considering the technical feasibility and applicability in the real-world environments, we think both of the approaches are equally important, although perhaps the second approach is of more immediate importance, whereas the first one is of more long-term significance. In this thesis, we will concentrate on the second approach, with an objective to develop a fuzzy processing system which can be added to an existing database management system (ORACLE) to make its processing of imprecise queries become immediately possible.

5. How to express and evaluate efficiently the membeship functions is another problem with database query using fuzzy sets. Theoretically, all records in a database satisfy any of fuzzy conditions, albeit at different degree of satisfaction (some may have a zero-degree). Thus, if a fuzzy query is not properly parsed, it is possible that a great number of records may be regarded as possibly qualified solutions and need to be retireved from the database, which will incur a heavy burden to the computer system. Mansfield and Fleischman [29] have described some approaches aiming to solve this problem, like building a library to store the commonly used membership functions, using trapezoid points to describe the support of a fuzzy set, restricting

the membership functions to piecewise linear functions, etc. Undoubtedly, a proper expression of a membership function is important, since it not only affects greatly the transaction processing efficiency, but also represents the meaning of the fuzzy term from the viewpoint of the user concerned. Considering these requirements, in this thesis we will explore the technique of using external functions as a means of expressing membership functions. This will provides the needed flexibilty for the users. A user whose main concern is the accuracy of the meaning of his fuzzy terms may choose to use some more appropriate functions as the membership functions, whereas a user whose main concern is the transaction time may choose to use some simple functions like piecewise linear functions to express his membership functions. Besides, in our system, a Parser and a Data Manager will be designed aiming to parse and process the membership functions efficiently.

6. Usually, a query may contain numerous fuzzy conditions. However, a user rarely regards all of his conditions to have equal importance. Therefore, operators applied to do the combination of the conditions become very important. Hopefully, these operators should reflect the viewpoint of the user on the combination of his conditions. For this purpose, a simple and common approach suggested is to associate different conditions with different weights. Wong and Leung [42] have attempted to build up an equivalence relationship between the problem of fuzzy query and the problem of multicriteria decision making. This is naturally an approach which may accommodate the viewpoints of the users who want to search for candidates from databases under multiple criteria. In this thesis, we will tackle the combination problem of fuzzy conditions from a different perspective. We will model the problem by the theory of decision making under uncertainties (multiple futures). This results in an approach which not only gives Wong and Leung's operators some new explanation, but also provides a model to better reflect the viewpoints of users who want to search for the best candidates under some uncertain environments.

Chapter 3

A FUZZY DATABASE QUERY LANGUAGE

In this Chapter we describe our query language to process fuzzy conditions contained in database queries, which is mainly based on the theory of fuzzy sets. As a preliminary, the following basic concepts are briefly introduced and reviewed first.

3.1 Basic Concepts of Fuzzy Sets

A set in mathematics has crisp boundaries, which is used to formally represent a precise concept. For example, the "integer numbers that are greater than 1 and less than 10" may be represented by the set $A = \{2, 3, 4, 5, 6, 7, 8, 9\}$ or by its characteristic function $\phi_A : X \rightarrow \{0, 1\}$, where X is the set of all integer numbers, $\phi_A(x) = 0$ means $x \notin A$ while $\phi_A(\mathrm{x}) = 1$ means $x \in A$.

In contrast to this classical set concept, a fuzzy set is a class of objects in which there are no sharp boundaries between those objects that belong to this class and those that do not. More precisely, we have

Definition 1 Let $X = \{x\}$ be a collection of objects. A *fuzzy set* $A \subset X$ is a set of pairs

$$A = (x, \mu_A(x)), x \in X \tag{3.1}$$

where $\mu_A(x)$ is termed the degree of membership of x, and $\mu_A : X \rightarrow M = [0,1]$ is a *membership* (characteristic) *function* from X to the interval M.

$\mu_A(x)$ is a function indicating the degree of membership of x in the set A. Roughly, $\mu_A(x) = 0$ means that x does not belong to A, $\mu_A(x) = 1$ means that x belongs to A, while $0 < \mu_A(x) < 1$ means that x partially belongs to A. (Exactly every x belongs to A, if $x \in X$, with varying degree of belongingness as given by the membership function $\mu_A(x)$). An ordinary set in the classical mathematical sense is a special case of a fuzzy set whose membership function can take only two values 0 and 1. Sometimes the pair $(x, \mu_A(x))$ is also denoted by $\mu_A(x)/x$.

A fuzzy set can be used to represent a fuzzy concept. An example is as follows.

Example Let X be the set of integers. The statement "several objects" may be represented by the following fuzzy set:

$$A = \{(3,0.4),(4,0.6),(5,1.0),(6,1.0),(7,0.8),(8,0.6)\} \cup \{(x,0.0) : x < 3, x > 8, x \in X\}$$

Some basic definitions are given below:

Emptiness A fuzzy set A is *empty* iff $\mu_A(x) \equiv 0$.

Normality A fuzzy set A is *normal* iff $sup_x \mu_A(x) = 1$. A fuzzy set is *subnormal* if it is not normal.

A non-empty subnormal fuzzy set can be normalized by dividing each $\mu_A(x)$ by a factor $sup_x \mu_A(x)$.

Support The *support* of a fuzzy set A is a set $S(A)$ such that $\mu_A(x) \geq 0 \Leftrightarrow x \in S(A)$. If $\mu_A(x) = constant$, $\forall x \in S(A)$, then A is said to be *nonfuzzy*.

Equality Two fuzzy sets A and B are *equal*, written as $A = B$, iff $\mu_A(x) = \mu_B(x), \forall x \in X$.

Containment A fuzzy set A is *contained in* or is a *subset* of a fuzzy set B, written as $A \subset B$, iff $\mu_A(x) \leq \mu_B(x)$.

Complementation A' is the *complement* of A iff $\mu_{A'}(x) = 1 - \mu_A(x)$.

Example. The fuzzy sets A = {large numbers} and A' = {not large number} are complements of one another.

Intersection The *intersection* of two fuzzy sets A and B is defined as the largest fuzzy set *contained in both* A and B and is denoted by $A \cap B$. The membership function of $A \cap B$ is given by $\mu_{A \cap B}(x) = min(\mu_A(x), \mu_B(x)), x \in X$, written as $\mu_{A \cap B} = \mu_A \wedge \mu_B$.

Remarks. Intersection bears a close relation to the connective "and". For example, if A = {tall men} and B = {fat men}, then $A \cap B$ = {tall and fat men}. The concept of intersection will be used in the operator AND in the Fuzzy Processor to be described below.

Union The union of two fuzzy sets A and B is defined as the smallest fuzzy set *containing both* A and B and is denoted by $A \cup B$. The membership function of $A \cup B$ is given by $\mu_{A \cup B}(x) = max(\mu_A(x), \mu_B(x)), x \in X$, written as $\mu_{A \cup B} = \mu_A \vee \mu_B$.

Remarks. Union bears a close relation to the connective "or". For example, if A = {tall men} and B = {fat men}, then $A \cup B$ = {tall or fat men}. The concept of union will be used in the operator OR in the Fuzzy Processor to be described below.

Algebraic sum The *algebraic sum* of two fuzzy sets A and B is denoted by $A \oplus B$ and is defined by $\mu_{A \oplus B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x), x \in X$.

Remarks. The concept of algebraic sum will be used in the operator COMB in the Fuzzy Processor to be described below.

Algebraic product The *algebraic product* of two fuzzy sets A and B is denoted by AB and is defined by $\mu_{AB}(x) = \mu_A(x)\mu_B(x), x \in X$.

Convex combination The *convex combination* of the fuzzy sets A, B, and Z is denoted by $(A, B; Z)$ and is defined by $(A, B; Z) = ZA + Z'B$, where Z' is the complement of Z, which is characterized by $\mu_{(A,B;Z)}(x) = \mu_Z(x)\mu_Z(x) + [1 - \mu_Z(x)]\mu_B(x), x \in X$.

Remarks. The concept of convex combination will be used in the operator HURWICZ in the Fuzzy Processor to be developed below.

Relation A *fuzzy relation* R in a product space $X = X_1 \times X_2 \times \cdots \times X_n$ is a fuzzy set in X characterized by an n-variate membership function $\mu_R(x_1, x_2, \cdots, x_n), x_i \in X_i, i = 1, 2, ..., n$.

Example. Let $X = Y = R^1$, where R^1 is the real line $(-\infty, +\infty)$. Then x $\gg$ y is a fuzzy relation in R^2, which may have $\mu_R(x, y) = 0$ for $x \leq y$ and $\mu_R(x, y) = \frac{1}{1+\frac{1}{(x-y)^6}}$ for $x > y$.

The above are some very basic concepts involving fuzzy sets which are related to our work in the sequel.

3.2 The Syntax of the Fuzzy Query Language

The syntax of our fuzzy query language is defined in extended BNF grammar, which is summarized as follows:

CONCEPT-DEF	::=	<*concept*> = CONDITION ;
QUERY	::=	select ITEM-LIST
		from RELATION-LIST
		where CONDITION ;
ITEM-LIST	::=	ITEM ITEM-LIST-1*
ITEM-LIST-1	::=	, ITEM
ITEM	::=	EXPRESSION | USERFUNCTION
COLUMN	::=	<*relation-name*>.<*column*> |
		<*column*>
EXPRESSION	::=	<*value*> | COLUMN |
		(EXPRESSION) |
		EXPRESSION MATHE-OPERATOR EXPRESSION
MATHE-OPERATOR	::=	$+ \mid - \mid * \mid /$
USERFUNCTION	::=	<*function*>FUNCTIONPAR
FUNCTIONPAR	::=	()| (PARAMETERS)
PARAMETERS	::=	ITEM PARA-1*
PARA-1	::=	, ITEM
RELATION-LIST	::=	RELATION RELATION-LIST-1*
RELATION-LIST-1	::=	, RELATION
RELATION	::=	<*relation-name*> | <*relation-name*> <*alias-name*>
CONDITION	::=	CONDITION-EXPRESSION |
		CONDITION-OPERATOR
		(CONDITION CONDITION-1$^+$)

CONDITION-1	::=	, CONDITION
CONDITION-EXPRESSION	::=	FUZ-EXPRESSION |
		NON-FUZ-EXPRESSION |
		<*concept*>
CONDITION-OPERATOR	::=	and | or | comb | poll | regret
		| hurwicz
FUZ-EXPRESSION	::=	ITEM is ((VALUE-LIST) (DEGREE-LIST))
		| USERFUNCTION
VALUE-LIST	::=	<*value*> <*value*>$^+$
DEGREE-LIST	::=	<*degree*> <*degree*>$^+$
NON-FUZ-EXPRESSION	::=	ITEM RELATIONAL-OPERATOR ITEM
RELATIONAL-OPERATOR	::=	< | > | = | <= | >= | <>

where

<*column*> is the name of a field in the database.

<*relation-name*> is the name of relation in the database.

<*value*> is the value of the type corresponding to a field.

<*degree*> is the value of the corresponding degree of membership.

<*concept*> is the words of users want to query.

<*function*> is the name of user function.

<*number*> is the real number.

The following are some examples:

1. If a user wants to retrieve the id and names of those companies whose business are marketing, his query should be stated as follows:

```
select co_id,co_name
from company
where business = 'Marketing'
```

where `company` is a relation table name, `co_id, co_name, business` are the column name of the table `company`, and `business = 'Marketing'` is a crispy condition.

2. If a user wants to retrieve the names and ages of those applicants who are male and whose height is taller than 178, the query should be as follows:

```
select applic_name,age
from applicant
where AND ( sex = 'M', height > 178 )
```

where `applicant` is a relation table name, `applic_name, age, sex, height` are the column names of the table `applicant`, and `sex='M'` and `height>178` are crispy conditions linked by the operator AND.

3. If a user wants to retrieve data on the student names, ages, average ages and average heights under the conditions that they are healthy and have good grade. The query can be stated as follows:

```
select name, age, avg(height), avg(age)
from student
where AND ( healthy, GPA is ((1 2 3 4)(0.2 0.4 0.7 1.0)) )
```

where `student` is a relation table name, `healthy` is a fuzzy concept, `GPA is ((1 2 3 4)(0.2 0.4 0.7 1.0))` is a fuzzy condition, and `avg(age), avg(height)` are external functions. Note that the fuzzy condition is `good GPA`, which is expressed in the query by two lists of data. The first list `(1 2 3 4)` contains the domain values of `GPA`, whereas the second list `(0.2 0.4 0.7 1.0)` gives the corresponding degrees of membership in terms of satisfying the fuzzy condition.

3.3 Fuzzy Operators

In general, a user's query may contain a number of conditions, expressed in the following form:

QUERY ::= select ITEM-LIST
from RELATION-LIST
where CONDITION.

where

CONDITION ::= CONDITION-EXPRESSION |
CONDITION-OPERATOR
(CONDITION CONDITION-1$^+$)

which is a combination of all crispy and fuzzy conditions of the user. The problem now is how to obtain the required data to satisfy these conditions.

It is clear that, under each fuzzy condition, there could be a set of candidates in the database, which all satisfy the condition in various degrees. An example is as follows, in which a user wants to find the names of those students in the Journalism Department who have good GPA and good sports_grade:

```
SELECT name
FROM student
WHERE AND (dept = 'Journalism',
GPA is ( ( 0.0 0.4 0.8 1.2 1.6 2.4 2.8 3.2 3.6 4.0)
( 0.0 0.0 0.0 0.0 0.0 0.0 0.2 0.6 0.9 1.0) ),
sports_grade is ((0.0 0.4 0.8 1.2 1.6 2.4 2.8 3.2 3.6 4.0)
(0.0 0.0 0.0 0.0 0.0 0.0 0.2 0.6 0.9 1.0 )) )
```

Assume that the names, GPAs and sports_grades of those students who satisfy the crispy condition (`dept = 'Journalism'`) have been retrieved from the database, which are given in the Table 3.1.

Accordingly, the degrees that the GPA and the Sports-Grade of each candidate are good can be computed by mapping, respectively, his/her GPA and the Sports-Grade to the

Table 3.1: Student Grades

Name	GPA	Sports-grade
Paul	3.80	2.60
Jimmy	2.60	3.60
Aaron	2.80	2.80
Boris	3.15	2.90
Gill	3.27	2.60
Felix	2.50	3.47
Galen	3.80	2.70
Lorna	3.40	3.33
Lyle	2.95	3.05
Nathan	3.20	2.60
Olive	3.47	2.80
Cora	3.33	3.33

membership distributions as given in the query. Let us illustrate the mapping mechanism by considering an example, the GPA of Paul, which equals 3.8. From the membership distribution in the query 'GPA is ((0.0 0.4 0.8 1.2 1.6 2.4 2.8 3.2 3.6 4.0) (0.0 0.0 0.0 0.0 0.0 0.0 0.2 0.6 0.9 1.0))', we see that GPA=3.8 lies in between 3.6 and 4.0. Using a linear interpolation with the two known points (3.6, 0.9) and (4.0, 1.0), we can get the degree of membership for GPA=3.8, which is 0.95. Using this method, we obtain Table 3.2.

In this example the problem is to rank the candidates according to their degrees of satisfying the two fuzzy conditions.

In general, when there are more than one fuzzy conditions contained in a query, we will face the problem of performing fuzzy set operations so as to determine the degrees of satisfying all the fuzzy conditions by the candidates. To do this, in our system six kinds of fuzzy operators are incorporated, which will be applied according to the need of users. These operators are now described below. Note that the first four operators were proposed by Wong and Leung in [42], whereas the last two are developed in the present project as their complement.

Table 3.2: Membership Degrees of Good Students

Name	Degree of Good GPA	Degree of Good Sports-grade
Paul	0.95	0.10
Jimmy	0.10	0.90
Aaron	0.20	0.20
Boris	0.55	0.30
Gill	0.65	0.10
Felix	0.05	0.80
Galen	0.95	0.15
Lorna	0.75	0.70
Lyle	0.35	0.45
Nathan	0.60	0.10
Olive	0.80	0.20
Cora	0.70	0.70

3.3.1 AND

This operator is to perform an intersection operation over the two fuzzy sets A and B corresponding to two fuzzy conditions,where $\mu_{A\cap B} = \mu_A \wedge \mu_B$. According to Section 2.1, this is equivalent to finding the minimum degree of membership, namely:

$$\mu_{A\cap B} = \min(\mu_A, \mu_B)$$

This rule can be extended to the general case where there are n fuzzy conditions (Let f_i^n denote the degree of membership of candidate i in the intersection set, and $\mu_{ij}, j = 1, \ldots, n$ be the degree of membership of candidate i under fuzzy condition j):

$$f_i^n = \min(\mu_{i1}, \mu_{i2}, ..., \mu_{in}) \ , \ i = 1, 2, ..., m.$$

where m is total number of candidates.

3.3.2 OR

This operator is to perform a union operation over the two fuzzy sets A and B (corresponding to two fuzzy conditions) where: $\mu_{A\cup B} = \mu_A \vee \mu_B$. According to the definition of union, this is equivalent to finding the maximum degree of membership, namely:

$$\mu_{A \cup B} = \max(\mu_A, \mu_B)$$

The extension of this rule to the general case with n fuzzy conditions (Let f_i^n denote the degree of membership of candidate i in the union set, and $\mu_{ij}, j = 1, \ldots, n$ be the degree of membership of candidate i under fuzzy condition j) is as follows:

$$f_i^n = \max(\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}) \ , \ i = 1, 2, ..., m.$$

3.3.3 COMB

This operator is actually to perform an algebraic sum operation over the two fuzzy sets A and B to get:

$$\mu_{A \oplus B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$$

In general, if there are n fuzzy conditions, the membership function of the fuzzy set after the algebraic sum operation is as follows ($i = 1, 2, ..., m$):

$$\begin{aligned} f_i^n(\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}) &= \mu_{i1}, && \text{if } n = 1; \\ f_i^n(\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}) &= \mu_{i1} + \mu_{i2} - \mu_{i1}\mu_{i2}, && \text{if } n = 2; \\ f_i^n(\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}) &= f_i^2(f_i^{n-1}(\mu_{i1}, \mu_{i2}, \ldots, \mu_{i,n-1}), \mu_{in}), && \text{otherwise.} \end{aligned}$$

3.3.4 POLL

This operator is simply to take the average degree of membership of the fuzzy conditions, namely, $f_i^2(\mu_{i1}, \mu_{i2}) = 1/2(\mu_A + \mu_B)$ for the example above. In the general case:

$$f_i^n(\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}) = \frac{1}{n} \sum_{j=1}^{n} \mu_{ij}, \quad i = 1, 2, \ldots, m.$$

•**Discussions:** The four operators described above were developed by Wong and Leung [42] based on the multicriteria decision-making theory. In general, suppose that there are n fuzzy conditions and m possibly qualified candidates retrieved from the database. See the table shown as follow. Let μ_{ij} be the degree that a candidate CA_i satisfies fuzzy condition FC_j.

	FC_1	FC_2	$\cdots$	$\cdots$	FC_n
CA_1	μ_{11}	μ_{12}	$\cdots$	$\cdots$	μ_{1n}
CA_2	μ_{21}	μ_{22}	$\cdots$	$\cdots$	μ_{2n}
$\vdots$					$\vdots$
$\vdots$					$\vdots$
CA_m	μ_{m1}	μ_{m2}	$\cdots$	$\cdots$	μ_{mn}

In this matrix FC_j can be regarded as the jth criterion of the decision maker. CA_i is his ith possible choice and μ_{ij} is the payoff under criterion FC_j by choosing CA_i. Then, according to the objective of the problem (like maximizing all the n criteria or maximizing one of the criteria), the decision maker will apply an appropriate operator (like **AND** or **OR**). This corresponds to our data retrieval problem in the following sense: if the user wants to find candidates with maximum degree of membership to satisfy all the fuzzy conditions, then **AND** should be applied, otherwise if he wants to find candidates with maximum degree of membership to satisfy one of the fuzzy conditions, then **OR** should be used. Similarly, the operator **POLL** should be used when each fuzzy condition is associated with an equal weight and the problem is to maximize their sum. The operator **COMB** should be used if the situation where the effect of applying all the attributes is less than the sum of the effect of each attribute used separately. For more detailed explanation on these operators, see [42].

The following two operators will be presented from a different perspective. Clearly there are situations where a user wants to determine some optimal options against certain unknown futures. For example, one may wish to determine what kinds of shares he should buy in a changing stock market. He knows that n possible futures, FC_j, $j = 1, 2, ...n$, may occur, and he wants to retrieve from a database those candidates that may maximize the expected payoff under the the uncertain futures. This problem can be formulated by the matrix above, where the column μ_{ij} represent the performance of different shares CA_i in a market condition FC_j. In such a situation, the retrieval problem becomes a problem of decision making under multiple futures.

It is easy to show that the operators **AND**, **OR** and **POLL** are equivalent to the decision-making rules **MAXIMIN**, **MAXIMAX**, and **LAPLACE** in decision making under multiple futures (cf. [33]). The next two operators **HURWICZ** and **REGRET** are generalized from the decision making theory under multiple futures based on the observation discussed above.

3.3.5 HURWICZ

This is a rule which performs a compromise between the rule **MAXIMAX** with extreme optimistic attitude and the rule **MAXIMIN** with extreme pessimistic attitude. It introduces an index of pessimism $0 \leq \alpha \leq 1$ to combine the rules **MAXIMAX** and **MAXMIN** (namely, **AND** and **OR**). Specifically, for our purpose, let us introduce the following fuzzy sets:

$$X = FC_1 \cap FC_2 \cap \ldots, \cap FC_n$$
$$Y = FC_1 \cup FC_2 \cup \ldots, \cup FC_n$$

and a fuzzy set Z characterized by $\mu_Z(x) = \alpha$ for all $x \in S(Z)$. Then, the HURWICZ operator is the convex combination of the fuzzy sets X, Y, and Z, namely:

$$(X, Y, Z) = ZX + Z'Y$$

which implies that

$$f^n = \mu_{(X,Y,Z)} = \alpha\mu_X + (1 - \alpha)\mu_Y$$

The selection of the index α depends upon the attitude of the user towards the evidence given by the best degree of membership and the worst degree of membership for an alternative. In fact, the final solution obtained by **HURWICZ** operator is a function of α, which can be plotted as a graph, enabling one to see a clearer picture on the change of the decision when the attitude of the user changes.

3.3.6 REGRET

This operator suggests that what we might really worry about is how bad we might feel afterwards when we see what we might have selected if we had known enough to do the right thing. Under such an attitude, a "regret" matrix has to be computed from the original payoff matrix by examining each column in turn.

Generally, suppose that there are n fuzzy conditions and m possibly qualified candidates retrieved from the database. Let us introduce the fuzzy set $\widetilde{FC}_j$ in which the degree of membership of the ith member is defined as $\tilde{\mu}_{ij} = \sup_i\{\mu_{ij}\} - \mu_{ij}$, for $j = 1, 2, \ldots, n$. (Note that $\widetilde{FC}_j$ becomes FC'_j if FC_j is normal, namely, $\sup_i\{\mu_{ij}\} = 1$). A regret matrix is one with $\tilde{\mu}_{ij}$ as its elements, where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$.

The operator **REGRET** is actually to perform an **OR** operation over $\widetilde{FC}_j, j = 1, 2, \ldots, n$, to obtain

$$X = \widetilde{FC}_1 \cup \widetilde{FC}_2 \cup \ldots \cup \widetilde{FC}_n$$

the degree of membership of its ith member is given by:

$$\tilde{f}_i^n = \max(\tilde{\mu}_{i1}, \tilde{\mu}_{i2}, \ldots, \tilde{\mu}_{in})$$

$\tilde{f}_i^n$ is the regret of candidate i, $i = 1, 2, \ldots, n$. If the candidates are to be ranked in terms of their suitability to satisfy the query conditions, they should be ranked in an increasing order of $\tilde{f}_i^n$, under the operator **REGRET**.

The application of the operator **REGRET** to the example presented in the beginning of the Section is given below. First the Regret matrix as shown in Table 3.3 is obtained. Then, applying the operator we get the result as in the Table 3.4:

Table 3.3: Regret Degree

Name	Regret(Grade)	Regret(Sports-grade)
Paul	0.00	0.80
Jimmy	0.85	0.00
Aaron	0.75	0.70
Boris	0.40	0.60
Gill	0.30	0.80
Felix	0.90	0.10
Galen	0.00	0.75
Lorna	0.20	0.20
Lyle	0.60	0.45
Nathan	0.35	0.80
Olive	0.15	0.70
Cora	0.25	0.20

Table 3.4: Maximum Regret

Name	Maximum Regret
Paul	0.80
Jimmy	0.85
Aaron	0.75
Boris	0.60
Gill	0.80
Felix	0.90
Galen	0.75
Lorna	0.20
Lyle	0.60
Nathan	0.80
Olive	0.70
Cora	0.25

Table 3.5: Six Operators

Name	AND	OR	COMB	POLL	HURWICZ	REGRET
Paul	0.100	0.950	0.955	0.525	0.695	0.800
Jimmy	0.100	0.900	0.910	0.500	0.660	0.850
Aaron	0.200	0.200	0.360	0.200	0.200	0.750
Boris	0.300	0.550	0.685	0.425	0.475	0.600
Gill	0.100	0.650	0.685	0.375	0.485	0.800
Felix	0.050	0.800	0.810	0.425	0.575	0.900
Galen	0.150	0.950	0.958	0.550	0.710	0.750
Lorna	0.700	0.750	0.925	0.725	0.735	0.200
Lyle	0.350	0.450	0.643	0.395	0.420	0.600
Nathan	0.100	0.600	0.640	0.350	0.450	0.800
Olive	0.200	0.800	0.840	0.500	0.620	0.700
Cora	0.700	0.700	0.910	0.700	0.700	0.250

In summary, the applications of the six operators described above to the example yield the results as given in Table 3.5.

Table 3.6 gives the rank of each candidate under each of the six operators:

Table 3.6: Comparison the six operators

Name	AND	OR	COMB	POLL	HURWICZ	REGRET
Paul	8	1	2	4	4	8
Jimmy	8	3	4	5	5	11
Aaron	5	12	12	12	12	6
Boris	4	10	8	7	9	3
Gill	8	8	8	10	8	8
Felix	12	4	7	7	7	12
Galen	7	1	1	3	2	6
Lorna	1	6	3	1	1	1
Lyle	3	11	10	9	11	3
Nathan	8	9	11	11	10	8
Olive	5	4	6	5	6	5
Cora	1	7	4	2	3	2

Chapter 4

SYSTEM DESIGN

In this Chapter, we shall describe in details the design of our system. Our main considerations, ideas, and approaches in designing the system so as to achieve its expected capabilities will be introduced. To do this, the organization of this Chapter is as follows: In the following Section we shall first discuss some main issues related to the general requirements of the system, such as the requirements on dealing with fuzzy conditions, highly aggregated concepts, and external functions; Our approaches of how to represent membership functions to define fuzzy conditions will also be introduced. Then, we shall introduce the overall architecture of the system, to show the various modules and the inter-relationships between these modules. The details of the various modules, including their expected functions, and our ideas and approaches to achieve these functions, will then be elaborated in the subsequent Sections respectively. These include the descriptions of the modules: Interface, Knowledge Base, Parser, ORACLE, Data Manager, and Fuzzy Processor.

4.1 General Requirements and Definitions

4.1.1 Requirements of the system

The main requeirements of the system are that it be able to handle various situations where user queries may contain highly aggregated concepts, fuzzy conditions, and external functions. Specifically, we expect that our system can deal with queries that involve:

Fuzzy conditions In traditional database management systems, queries are intended to retrieve data to satisfy only crispy conditions. In many cases, this lack of flexibility leads to empty answers. That is one of the reasons why we want to investigate the extension of such systems so that they are able to support imprecise/fuzzy querying capabilities. To be distinguished from those highly aggregated concepts we shall discuss below, a fuzzy condition is defined by our system as one: (i) that can be represented by a membership function; and (ii) that involves only attributes whose values can be found in the database directly. An example is the condition `high GPA`, which can be expressed directly by "`GPA is ((1 2 3 4)(0.2 0.4 0.7 1.0))`" (see Section 3.2), and the values of the attribute GPA can be found in the database. Thus, when the GPA value of a candidate is obtained, the degree that this candidate satisfies the fuzzy condition can be evaluated by using the membership function. In our system, a `Fuzzy processor` will be designed and developed to handle the fuzzy conditions to entertain imprecise/fuzzy queries. The six fuzzy operators as described in Chapter 3 will be used in the Fuzzy Processor, to deal with the cases where numerous conditions are to be combined together.

Highly aggregated concepts A query may contain highly-aggregated concepts. In contrast to fuzzy conditions, such an aggregated concept is defined in our system as one that cannot be processed directly without being further decomposed into more specific crispy and/or fuzzy conditions. For example, consider the query "find a `suitable candidate` to fill the position of Computer Officer". In this query, `suitable candidate` is an aggregated concept, which is equivalent to the combination of a number of (crispy and/or fuzzy) conditions, such as: the candidate has

an advanced degree in computer science or a related area, high GPA, at least two years of supervisory experience, young, and so on. Clearly, a query involving aggregated concepts often occurs when a user is able to submit a general, vaguely-defined request only. Such a concept must be further decomposed into more specific crispy conditions, fuzzy conditions and subconcepts, before any candidate in the database can be evaluated as to whether he satisfies the requirement. The decomposition, however, usually needs the knowledge of experts on the concept, and an ordinary user may not have the knowledge necessary to decompose his concept into these specific conditions. A typical query of this nature is, for example, "find those shares which have `low risk` to loose in the coming year", a query usually asked by ordinary people when considering buying certain shares. Clearly, to answer this query, the knowledge of stock-market specialists must be applied to further identify the necessary conditions to define the concept `low risk`.

Our system will possess the capacity to handle queries involving aggregated concepts (for convenience, the term `aggregated concept` will be, from now on, abbreviated to `concept`). This is to be performed by a built-in knowledge base in the system. In the Knowledge Base, each concept will be represented by a concept tree, which describes the composition of the concept. Each concept tree will have a hierarchical structure, which gives different level of definitions on the concept. The most general concept is located at the highest level node whereas specific (crispy or fuzzy) conditions and subconcepts (each of which may also be defined by another concept tree) are located at the lower level nodes. Many concept trees are stored in the knowledge base, which are constructed based on the knowledge of experts on the concepts that may appear in queries related to the database under consideration. These concept trees can be inserted, modified and deleted at any time. A detailed description of our knowledge base will be given in Section 4.4 below.

External functions Because of the limitation of internal functions, it is desirable that functions can be defined by a user in accordance with his requirement. Our system

will be able to handle the case where a query contains external functions defined by a user in a file. Those functions can be modified and inserted if needed. They are compiled by a compiler in the system before they are called. We will use the technique of dynamic library to link the external functions. The use of external functions provides much more capabilities to the system. For example, we can use an external function to represent a membership function of a fuzzy condition. This leads to more flexible and accurate representation of membership functions. This will be elaborated below.

4.1.2 Representation of membership functions

Our system will apply the theory of fuzzy sets to handle fuzzy conditions involved in user queries, which will therefore need to define a fuzzy condition by using a membership function. Consequently, it is important to design some appropriate methods to represent membership functions in the system. In [42], a membership function is expressed by two lists. The first one contains domain values for a field in ascending or descending order. The second list gives the corresponding degrees of membership. For example, the fuzzy condition "old "(shown in Fig 4.1) is expressed as:

age is ((15 20 25 30 40 50 60 70)(0.2 0.3 0.4 0.5 0.7 0.8 0.9 1.0)).

We call this method of using two lists to represent a membership function the `primitive representative method`. The method has the advantage that it is easy to understand and implement. However, it becomes very cumbersome in some complicated situations, particularly when the membership function depends upon more than one variable. Consider a problem where a person wants to find a job near his home. He may thus present a fuzzy condition that the distance of the job location from the origin (representing his home) is *short*. Clearly, the degree of membership of a point in this fuzzy set is a function of two variables x and y in the plane. In such a case, it will be very difficult to enumerate all the possible combinations of x and y so as to give the two lists in the primitive

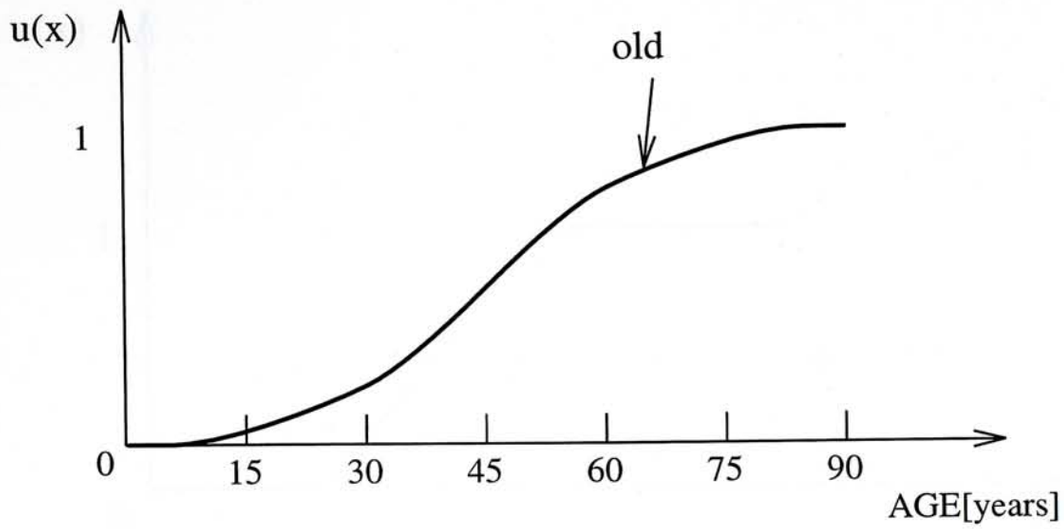


Figure 4.1: The membership function for the concept "old"

representative method. Besides, even in a case where there is only a single variable x, the primitive method is convenient only when the membership function is monotonously increasing or decreasing, see Fig 4.2 and Fig 4.3. In other cases, say, the membership function is convex or non-convex (see Fig 4.4 and Fig 4.5), the primitive representative method is inconvenient, which may need a great deal of points to completely represent the whole curve.

Our system will have the capability to provide two kinds of methods to represent membership functions, which are described below respectively:

Primitive Representation The system does not exclude the primitive representative method. In other words, a user can state, if he likes, a membership function by two lists as required by the primitive representative method. In such a case, our system will treat the membership function as a piece-wise linear function, which will use a straight line to connect every pair of adjacent points, called corner points, as given by the two lists. Then, when the degree of membership for a point in between a pair of corner points is needed to be evaluated, the linear function will be applied. This method is easy to realize, although it may introduce some error when the original function is a non-linear function and not sufficiently many points are provided by

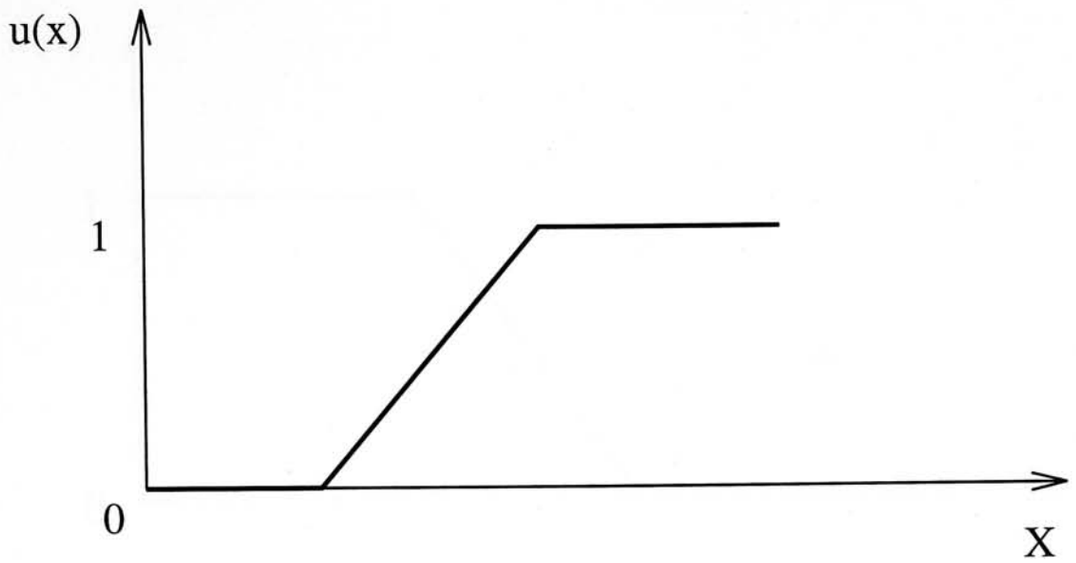


Figure 4.2: Increasing membership function

the user.

External Function A user can define a membership by using an external function. This is the most general way to do the representation, as the user can choose the most appropriate function to represent the membership. For example, consider a fuzzy condition that the working experience of a candidate should be `about eight years`. Suppose that, according to the opinion of the user, the membership function on this fuzzy condition should be a normal function, defined under the conditions that: (i) when the number of years of working experience of a candidate is exactly eight, then the degree that this candidate meets the condition is exactly 1; and (ii) when the number of years of working experience is greater than twelve or less than four, then the degree becomes zero. Our system uses an external function `normdist (minimum_value, maximum_value, value)` to represent this membership function, where the parameters `minimum_value` and `maximum_value` give the range of the distribution (equal to four and twelve respectively in the present example), and the function value is set to be one at the middle of the range. Therefore, whenever a value (the number of years) is given for the parameter `value`, the normdist()

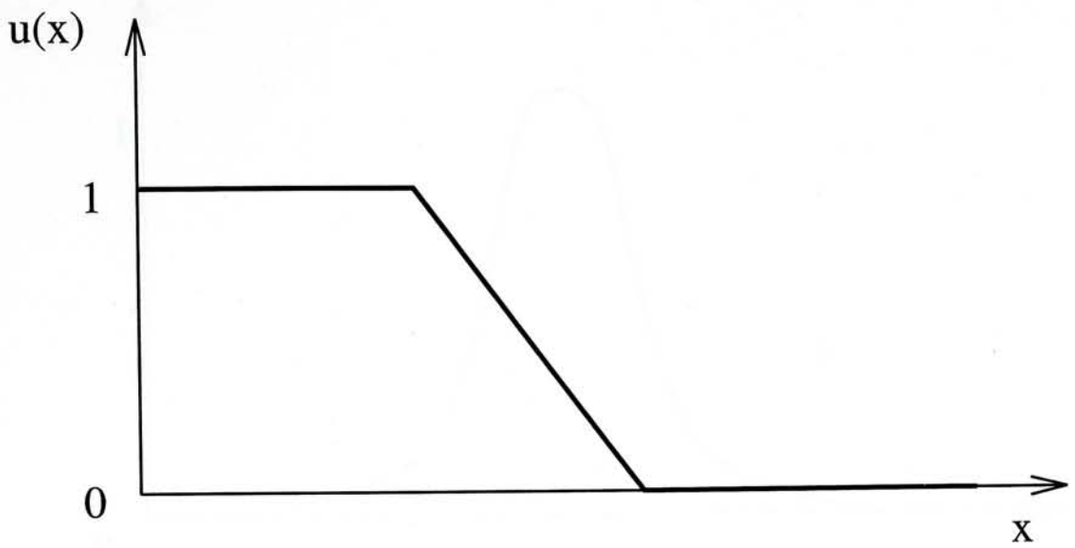


Figure 4.3: decreasing membership function

function returns the membership degree as required.

An external function in the system can be a function of more than one variable. This overcomes the difficulty of the primitive representative method. The use of external functions provides a general and accurate approach to the representation of membership functions. The implementation of this approach is feasible, as long as the system can accept and handle external functions. Our system has this expected capacity, as we have discussed above.

4.2 Overall Architecture

The overall architecture of the system is depicted in Fig 4.6.

The system consists of the following modules: **Interface**, **Knowledge Base**, **Parser**, **ORACLE**, **Data Manager**, and **Fuzzy Processor**. It supports a user query in the fuzzy query language (FQL) as described in Chapter 3. The FQL virtually comes from a hybrid of the fuzzy set theory and the decision-making theory, as we can see from Chapter 3. The query system is designed to retrieve data via a database management system – ORACLE running on SUN Sparc workstations. It issues queries written in SQL,

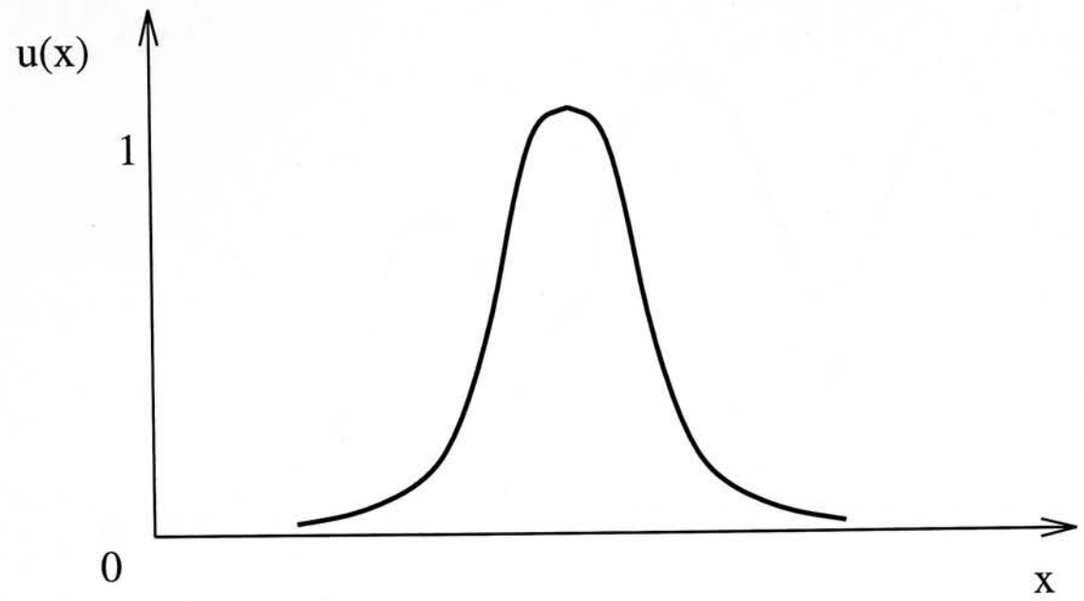


Figure 4.4: Convex fuzzy set

a set of commands that all programs must use to get access to data from the Database managed by ORACLE.

A user query may include fuzzy and crispy conditions, aggregated concepts, and external functions. The `ITEM_LIST`(see Section 3.2) of a query may be column names of tables and/or external functions. The mechanism of the system is that the fuzzy query will be translated to a SQL query first, which contains the crispy conditions appearing in the user query as well as some new crispy conditions extracted from the fuzzy conditions and concepts. Then, the data that satisfy all these crispy conditions are retrieved from the database via ORACLE using the SQL query. The system then processes those data to determine the degrees that they satisfy the fuzzy conditions. Specifically, the system works as follows (cf. Fig 4.6):

1. A user submits a query to retrieve data in accordance with his requirements to the **Interface** module. The Interface module passes the query to the Parser, and waits for the result.

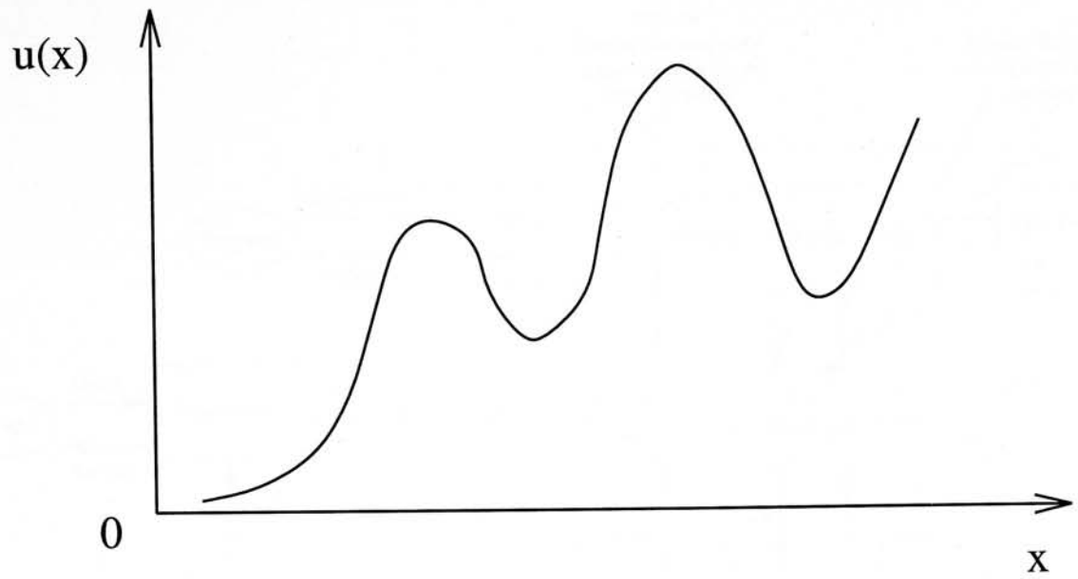


Figure 4.5: Nonconvex fuzzy set

2. If the query contains a general concept, parser will pass the concept to the **Knowledge Base**. The Knowledge Base will translate the concept into certain specific conditions in accordance with the knowledge on the concept in the Knowledge Base.

3. The **Parser** will pass all conditions, including those presented in the query and those generated by the Knowledge Base, to the Fuzzy Processor. On the other hand, the Parser will generate a number of crispy conditions corresponding to each fuzzy condition. These new crispy conditions will be combined with the original crispy conditions to generate a condition without fuzzy criteria. The Parser then eliminates the external functions in the query to generate a SQL query. Section 4.5 gives a detail description of the Parser.

4. The SQL query will be submitted to **ORACLE** to retrieve data from the database. Those data (candidates) satisfy all the crispy conditions (including those new cripsy conditions generated from the fuzzy conditions by the Parser) in the SQL query, and are potential candidates to the user.

5. For each potential candidates, the **Data Manager** evaluates its user functions, and

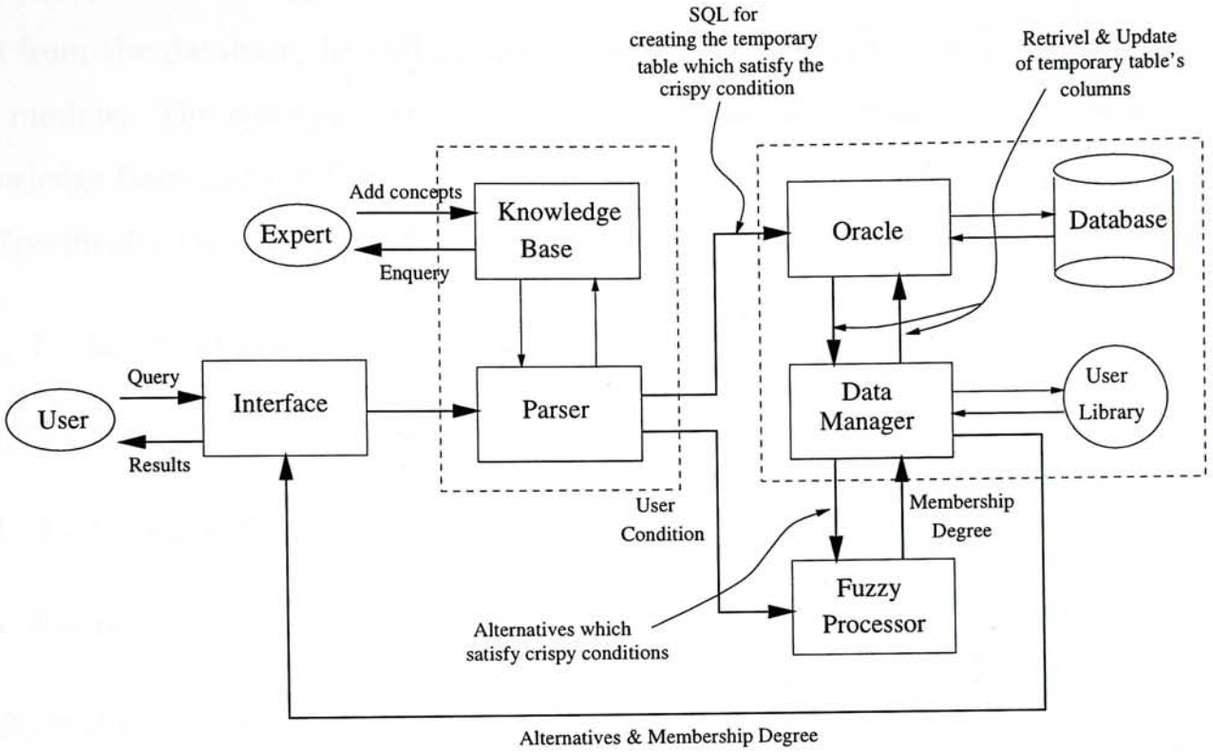


Figure 4.6: Architecture of Overall Design

then sends the candidate to the Fuzzy processor.

6. The **Fuzzy Processor** determines the degrees of membership that the candidates satisfy the user condition. The membership degree is returned to the Data Manager.

7. The Data Manager ranks the candidates with non-zero degrees of membership under the fuzzy conditions. It sends them to the **Interface** together with the values of the external functions.

8. The **Interface** module provides these final results to the user.

The above is an outline about how the system works. The functions of the modules as discussed above will be now described in details in the following Sections.

4.3 Interface

This module is designed to directly interact with users. If a user wants to retrieve data from the database, he will submit a query in accordance with his requirements to this module. The concepts and the conditions in his query will then be passed to the Knowledge Base and the Parser respectively.

Specifically, this module will firstly ask the user to provide the following information:

1. 1 – Input the query
2. 2 – Manipulate concept(s)
3. 3 – Initialize the knowledge base
4. 0 – Exit

Note that a user can only choice 1 to input his query. After he chooses 1, the system will ask for the following information:

1. Enter the select item names.
2. Enter the relation table names.
3. Enter the conditions.

An example is as follows, in which a user wants to retrieve the name, age, gpa, avgc(gpa, student) of students who are academically excellent and whose gpa is greater than the average gpa:

```
Enter the item-name you are interested :
name,age,gpa,avgc(gpa,student)
Enter the relation table name :
student
Enter the conditions :
```

```
AND(academically excellent, gpa>avgc(gpa,student) )
```

In the query, `academically excellent` is a concept, and `avgc(gpa,student)` is an external function. The general concept has to be further decomposed into more detailed conditions recognized by ORACLE. The function `avgc(gpa,student)`, like other external functions, is created in C language in the user library. The system will call the user library and pass the parameter to obtain the corresponding function value.

The query inputted by the user will be now passed to the Parser. Besides, if it contains general concepts, these concepts will be passed to the Knowledge Base for further processing.

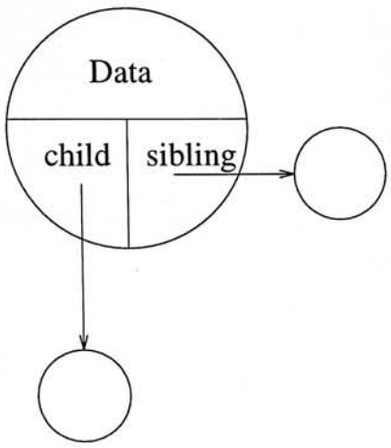
Another function of the Interface module is to receive the final result from the Fuzzy Processor (cf. Fig 4.6) and then send it to the user.

4.4 Knowledge Base

The Knowledge Base module will be activated when a general concept is contained in the query. The Knowledge Base module generates detailed, recognized conditions to define the general concept.

What is the knowledge base ? How is it represented ? In our knowledge base, a concept is represented by a condition tree. In a tree, a node may be a concept, an operator or a condition. Each node may have children and sibling. The data structure is shown in Fig 4.7, Fig 4.8 and Fig 4.9.

Note that in Fig 4.9, the tree on the left-hand-side is the representation of a concept, whereas the tree on the right-hand-side is its corresponding data structure. In this data structure, all the nodes at a same level are brothers, and the nodes at a lower level are the children of the node at the higher level.




```
typedef struct datatree {
 Token data;
  struct datatree *child, *sibling;
} Dtree;
```

Figure 4.7: Structure of Dtree

This is a hierarchical structure. The highest level node is a concept to be represented. The low level nodes are subconcepts or conditions. Each subconcept has its own concept tree which describes this subconcept. Thus, in the Knowledge Base, there are many small concept trees rather than just a single one. Concept trees with this kind of data struture are easy to be modified and can avoid redundance.

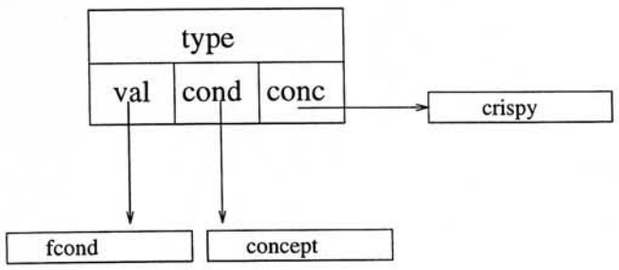
An example is as follows:

Suppose one wants to find the names of `good` sports students. Clearly `good` is a concept. The following is a concept tree in the knowledge base to define this concept:

```
good sports-student
 and
  term_gpa >= 3.0
  young
  tall
```

The above concept tree can also be contained in another concept, such as `excellent` sports students. The condition tree to define this concept is below:

```
excellent sports-student
 and
```




```
typedef enum{
 LEFTPAR,RIGHTPAR,AND,OR,
 COMB,POLL,REGRET,HUWICZ,
  FUZCOND,CONCEPT,NFUZCOND
} Tokentype ;

typedef struct {
  Tokentype type;
  Fcond *val;
  char *conc;
  char *cond;
} Token;
```

Figure 4.8: structure of Token

```
gpa >= 3.5
good sports-student
```

which will need the "good sports-student" tree as a subtree.

The fuzzy conditions "young" and "tall" can also be treated as concepts and defined in the knowledge base directly by the following conditions:

```
young
 age is ((40 35 30 25 20 15 ) (0.0 0.3 0.5 0.7 0.9 1.0) )


tall
 height is ( (150 160 170 180) (0.0 0.3 0.6 0.8 1.0) )
```

To locate a concept tree, we choose to use hashing index with the concept name as the key. Assume that we decide to have n buckets and define a hash function that maps the concept. Each bucket points to a chain which has data structure as follows:

```
typedef struct chain {
```

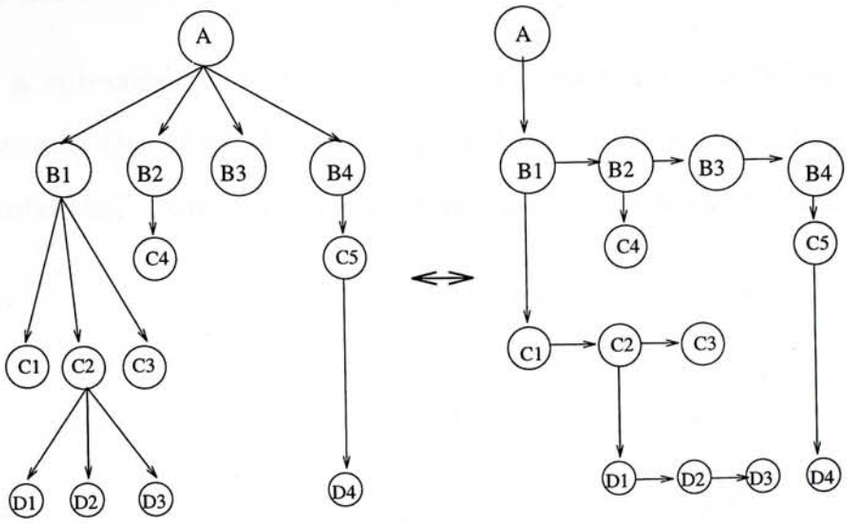


Figure 4.9: transformation

```
    Dtree *concept;
    struct chain *next;
} Chain;
```

The hash table is shown in Fig 4.10. Our approach works in the following way:

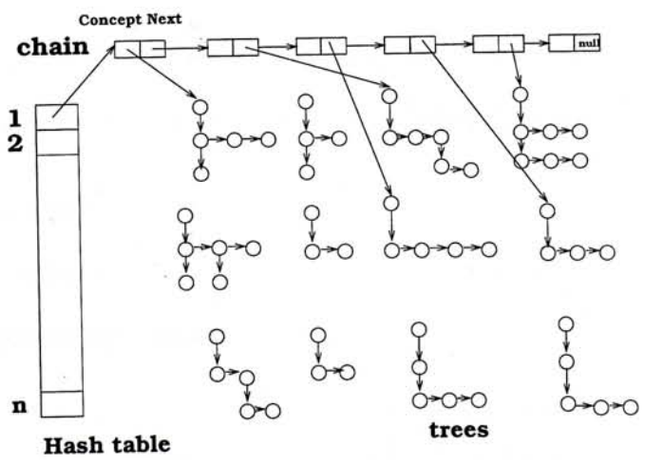


Figure 4.10: Hashing index

1. Firstly, if there is a concept in the query, we search this concept in the hash table,

and obtain the tree about this concept.

2. If there is a sub-concept in this tree, we will search the sub-concept in the hash table again, and then link the sub-concept tree to the concept tree. The process is repeated until there is no more concept but only well-defined conditions in the tree.

3. The condition in a query can be treated as a temporary concept. We build the condition tree for the query by the same mechanism. The whole condition tree generated is passed to the Parser module for further processing.

A concept tree can be inputted directly or can be read from a file that has already been saved. Specifically, the knowledge tree can be obtained by the following two methods:

1. Experts' knowledge on a concept can be collected and inputted by editing the relevant data file.

2. A user submitting a query involving a concept may be asked to interpret his understanding of the concept. He will be asked to define the conditions of the concept using the CONCEPT-DEF statement in the Interface module. The statement will be translated to a file of the standard format.

The Knowledge Base can be modified in any time by using expert knowledge. The interface of inputting concepts is shown by the following menu:

```
Please choose:
1)Add a concept
2)Load Knowledge Base
```

When we want to add a concept, we can choose 1, and then input concept name and conditions. The syntax is same as CONCEPT-DEF statement. When we want to add a file in the Knowledge Base, we should choose 2. The system will ask for the file name and then add the new file to Knowledge Base.

4.5 Parser

A Parser is designed in the system to process the crispy and fuzzy conditions that are expressed in the original query and those that are transferred from the Knowledge Base after interpreting the general concepts contained in the query. The Parser module will generate a SQL query to retrieve data from the database via ORACLE. Note that the system will not provide alternatives with a zero degree of membership for any fuzzy condition, the Parser will first find out the threshold point where the degree of membership of a fuzzy condition becomes zero beyond this point. The threshold point is used to construct a new crispy condition to limit the range of retrieval from the database. These new crispy conditions together with the original crispy conditions will generate a set of non-fuzzy conditions. If those non-fuzzy conditions contain external functions, they will be converted to conditions with no external functions. The non-fuzzy and non-functional conditions will be used in the intermediate SQL query. In summary, the Parser performs the following:

1. Examine the threshold points of the degrees of membership (where the degree of membership becomes zero) for each fuzzy condition. For each fuzzy condition, generate some new crispy conditions which prevent the system from unnecessarily retrieving alternatives with a zero degree of membership under this fuzzy condition. For example, from the fuzzy condition `young: age is ((45 40 35 30 25 20 15) (0.0 0.0 0.3 0.5 0.7 0.9 1.0))`, a crispy condition `age` $\leq$ `40` will be generated.

2. A query which contains two fuzzy conditions will have to specify how the combination of these conditions should be satisfied. This is to perform a set operation among the two sets of candidates retrieved under the two conditions, such as an "AND" or an "OR" operation. As we know, AND will consider the minimum degree of membership, whereas OR will consider the maximum degree of membership.

In a case where a user would prefer something in between these two extreme points, other operators will be used. In Chapter 3, we have described six operators, AND, OR, COMB, POLL, HURWICZ, and REGRET. In our system, a user may choose one of these operators to link each pair of fuzzy conditions according to his requirements. Note that a SQL command can only have AND and OR, the Parser will first convert the six fuzzy operators to one of AND and OR. Clearly, if the fuzzy operator is AND or OR, the corresponding operator in the SQL query will be the same. For the other four fuzzy operators, the Parser will perform the following: (1) pass them to the Fuzzy Processor module; and (2) use an OR operator to replace them when constructing the SQL query. The rational behind the second step is that all alternatives which satisfy any one of the pair of the fuzzy conditions should be first retrieved out from the database, which will then be sent to the Fuzzy Processor so that the Fuzzy Processor can use the original fuzzy operators to process them. This will guarantee that no qualified alternative is missed out from the retrieval.

3. Convert a query with external functions into one with no external functions. The technical details for the conversion will be described in Chapter 4.

4. Generate an intermediate SQL query using the original crispy conditions as well as the new crispy conditions, which will be issued to ORACLE to retrieve records from the database.

5. Pass all fuzzy conditions to the Fuzzy Processor and pass all external functions to the Data Manager. These two modules will further process those records retrieved from the database.

4.6 ORACLE

After the FQL (fuzzy query language) query submitted by the user is transformed to an intermediate SQL query with only crispy conditions and non-functional conditions,

they will be sent to the ORACLE. The alternatives which satisfies all the crispy conditions will be retrieved from the database through ORACLE and passed to the Data Manager. The Data Manager will evaluate, if necessary, the external functions and further pass the alternatives to the Fuzzy Processor to see to what degree an alternative will satisfy the fuzzy conditions.

4.7 Data Manager

This module is designed to handle user external functions given in a user query. When data are retrieved from the Database using the intermediate SQL query, they are not returned to the user directly. Instead, they will be put on a temporary table `Temp`, which will then be processed, by the Data Manager and the Fuzzy Processor, with consideration of the external functions as well as the fuzzy conditions before they become the final results to be sent to the user. The procedure is illustrated in Fig 4.11, which is described below:

1. In general, a user query contains three parts – item list, relation table and conditions, where the conditions may consist of fuzzy conditions and non-fuzzy conditions, which may contain external functions. The Parser will pre-process the query to generate an intermediate SQL query. Meanwhile, the conditions with external functions will be passed to the Data Manager and the fuzzy conditions will be passed to the Fuzzy Processor. Specifically, the operations involved on the **Item list, Table**, and **Conditions** are as follows:

   Item list Firstly, the Parser of the system will process item list. If an item list of the user query involves external functions, the Parser will generate a new item list for the intermediate SQL query. The new item list contains all columns and all parameters of the functions of the item list. For example, if the item list is

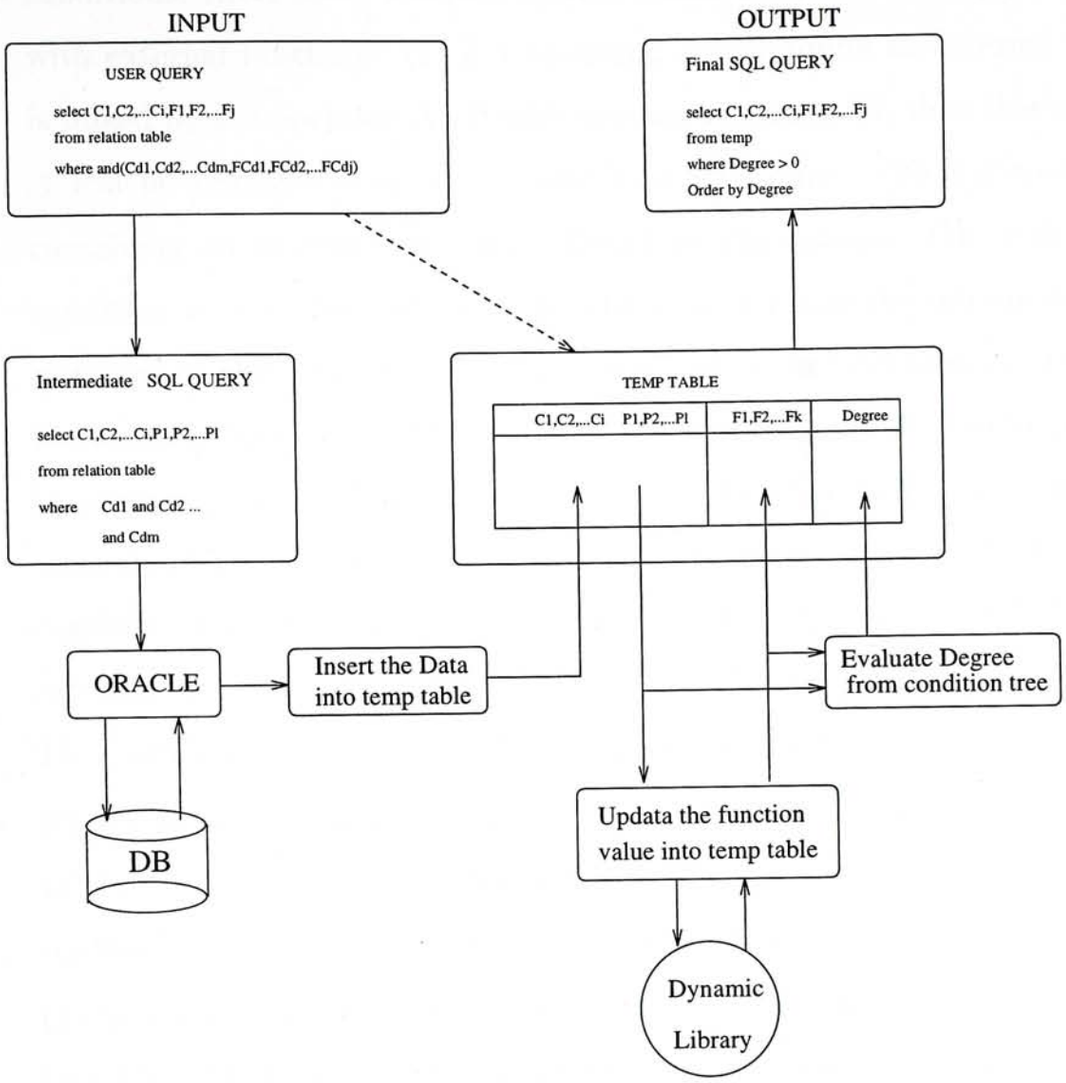


Figure 4.11: The diagram of calling external function

`name, age, height, substr(name,1,3), usrfunc4(gpa),`

then the Parser will generate the item list of the intermediate SQL query as

`name, age, height, 1, 3, gpa.`

Table The relation table name will be unchanged in the intermediate SQL query.

Conditions The Parser of the system uses the user conditions to generate new conditions for the intermediate SQL query. The new conditions should be non-fuzzy and contain no functions. We have illustrated, in the Section – Parser above, the method of how to generate new crispy conditions based on the fuzzy

conditions. After doing that, the system adopts the following approach to deal with external functions: (1) if a condition A containing an external function is linked by the operator AND with another condition B, then this condition A will be removed from the intermediate SQL query; (2) if a condition A containing an external function is linked by the operator OR with another condition B, then both A and B will be removed from the intermediate SQL query; and (3) after those records are retrieved by the intermediate SQL query, the external functions will be evaluated and the records will then be processed by applying the original conditions with the external functions. The reason to apply steps (1) and (2) above is to guarantee that no potentially qualified candidate is missed out from the retrieval by the intermediate SQL, before the external functions can be evaluated and thus the condition A can be checked. This can be justified by the following simple argument:

(1) Let the records contained in the Database satisfying the condition "A and B" be R1. A SQL query with only condition B will get those records that contain R1, since the intersection of A and B belongs to any one of A and B.

(2) Similarly, let the records contained in the Database satisfying the condition "A or B" be R2. A SQL query without conditions A and B will get those records that contain R2. This can be seen from Fig 4.12.

An example is shown below:

"`and(gpa is ((1.0 1.5 1.8 2.0 3.0 4.0)(0.0 0.0 0.2 0.6 0.8 1.0)), age>20, usrfunc2(height)>155,or (usrfunc3(age) > usrfunc4(gpa*10), age > 25 ) )`".

The Parser will pass the conditions with external functions and the fuzzy condition to the Data Manager and the Fuzzy Processor, and generate an intermediate SQL query with the following conditions:

"`and(gpa > 1.5, age > 20 )`".

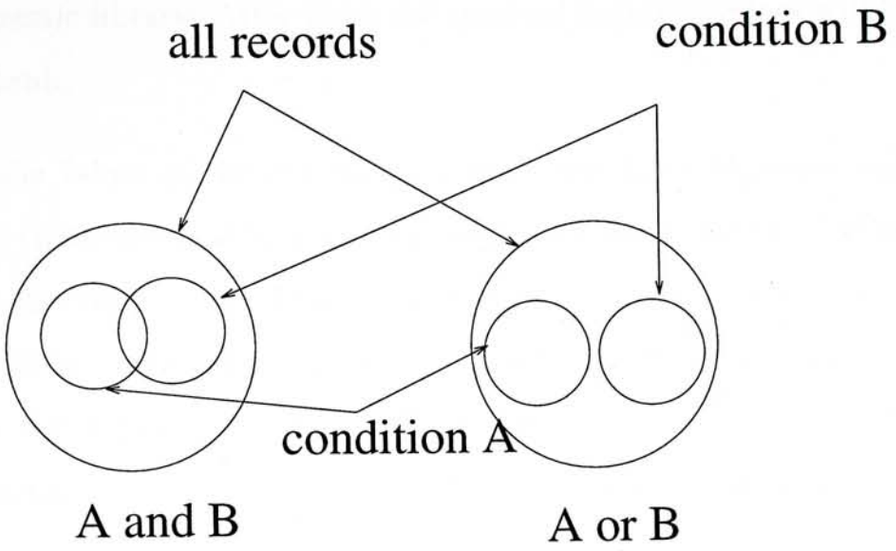


Figure 4.12: Intersection and union of A and B

Temp table The Data Manager creates a new `Temp` table based on the user query. The columns of `Temp` will consist of all columns, parameters and external functions which appear in the user query.

For example, if the user query's item list is :

```
name, age, height, substr(name,1,3), usrfunc4(gpa).
```

and the conditions are:

```
"and(gpa is ((1.0 1.5 1.8 2.0 3.0 4.0)(0.0 0.0 0.2 0.6 0.8 1.0)),
age>20, usrfunc2(height)>155,or (usrfunc3(age) > usrfunc4(gpa*10),
age > 25 ) )".
```

then the `Temp` table columns are

```
name, age, height, gpa, 1, 3, usrfunc2(height), usrfunc3(age),
usrfunc4(gpa*10), gpa*10, substr(name,1,3).
```

2. When the required data are retrieved from the Database by the intermediate SQL query, the Data Manager will insert these data into the `Temp` table. Then, it will evaluate the function values using these data by executing the external functions in

a user dynamic library. After that, the external function values will be updated in the `Temp` table.

3. Knowing the values of the external functions, the Data Manager will apply those conditions (with external functions) passed from the Parser to further update the records in the `Temp` table. Those records that do not satisfy the crispy conditions (if any) will be eliminated. After that, the Fuzzy Processor will be activated to determine the degrees of membership of the records in the `Temp` table under the fuzzy conditions. These degrees will be inserted in the `Temp` table.

4. The Data Manager will create a final SQL query. The item list of the final SQL query is same as the item list of the original user query. The relation table is `Temp` table. Since it is a requirement that the degree of membership of any record to satisfy the fuzzy conditions should be greater than zero, the final SQL query will carry a condition that only those candidates that have a non-zero degree of membership should be retrieved. The alternatives obtained by the final SQL query will be sorted and ranked according to their degrees of satisfying the fuzzy conditions, and then passed to the Interface module for transmission to the user.

A technique of dynamic library, after being enhanced to suit our need, is used to handle external functions. This will be described in details in Chapter 5.

4.8 Fuzzy Processor

The Fuzzy Processor receives information from two modules: one is Parser, which passes on the fuzzy conditions, and the other is the Data Manager, which passes on the records which have been processed with condition of the external functions. The Fuzzy Processor will apply the various operators available, such as AND, OR, COMB, PULL, HURWICZ, and REGRET, to calculate the degree of each record in terms of satisfying all the fuzzy conditions.

The functions of the fuzzy operators used in our system have been described in details in Chpater 2.

For example, we have a user query as follows :

```
SELECT name, age, height, usrfunc3(gpa), substr(name,1,3)
FROM student
WHERE and ( height>160, usrfunc3(age)>24,
      or (height is ((150 160 170 180) (0.6 0.7 0.8 1.0)),
           gpa > 3.5 ))
```

The system will generate the intermediate SQL query as:

```
SELECT name, age, height, gpa, 1, 3, gpa*10
FROM student
WHERE (height>160)
```

The columns of temp table are :

`name, age, height, 1, 3, gpa, usrfunc3(gpa), substr(name,1,3), usrfunc3(age)`

After the data of temp table are inserted and updated, the result will be obtained by the final SQL query as follows:

```
SELECT name, age, height, usrfunc3(gpa), substr(name,1,3)
FROM temp
WHERE degree > 0
ORDER BY degree
```

The name, age height,usrfunc3(gpa), substr(name,1,3) will be substituted by C0, C1, C2, F1 and F2 which are columns name in the `Temp` table.

Chapter 5

IMPLEMENTION

In Chapter 4, we have described the architecture and the main requirement and definition of our overall system. We have also presented in details the main functions of each module within the system so that the overall system can achieve its expected capacities. In this Chapter, we will describe some main considerations on the implementation of the system and its modules. We will highlight in the following Sections certain key issues in developing the Knowledge Base and the Data Manager, the two modules which require substantial programming effort.

5.1 Some General Considerations

1. At the begining, we considered choosing a language to program the system. The general requirement of this language is its capability to retrieve data from Database. We chose the C language, considering that it can embed SQL and that it is more portable. The C language can be supported by a wide variety of computers. It is less restrictive and more general than other languages, which makes it more convenient and effective for many complicated tasks.

2. We chose to use ORACLE, considering that it is a standard Relational Database Interface. ORACLE precompilers can support embeded SQL and dynamic SQL,

and can interpret embedded SQL statements and translate them into statements that can be understood by procedural language compilers. ORACLE tools support all features of ORACLE's SQL language.

3. We chose to use the Structured Query Language (SQL), as it provides a much needed common avenue of discourse between the end-user and programmer. It can be embedded by the programmer in procedural language such as C language. The SQL is more standard than the ORACLE CALL INTERFACE.

5.2 Knowledge Base

5.2.1 Converting a concept into conditions

As we have shown in Chapter 4, the main function of the Knowledge Base module is to process concepts contained in user queries. A concept will be transformed and represented by a hierarchical concept tree, and then converted by specific fuzzy conditions and crispy conditions in this module. This is carried out by the following procedure:

1. If the conditions in a user query contain a concept, the conditions will be transformed into a temp tree. In the program, the temp tree is represented by a file having a standard format. For example, suppose a sport institute is recruiting some students, under the conditions "tall" and "young". The "tall" is a concept. The "young" is represented by a fuzzy condition " age is ((60 50 40 30 20)(0 0.2 0.3 0.5 1.0))". The query will be:

```
SELECT name
FROM student
WHERE AND(tall,age is ((60 50 40 30 20)(0 0.2 0.3 0.5 1)) )
```

The conditions will be changed to a temp tree. See Fig 5.1

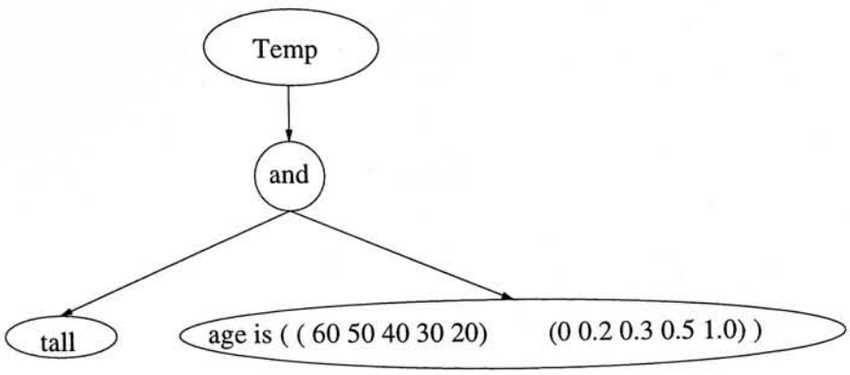


Figure 5.1: The temp tree

2. After the conditions are changed to a file, the system will search the concepts in the hash table.

   The "tall" is a concept. The system will search "tall" in the hash table of the Knowledge Base. The concept tree for "tall" shown in Fig 5.2.

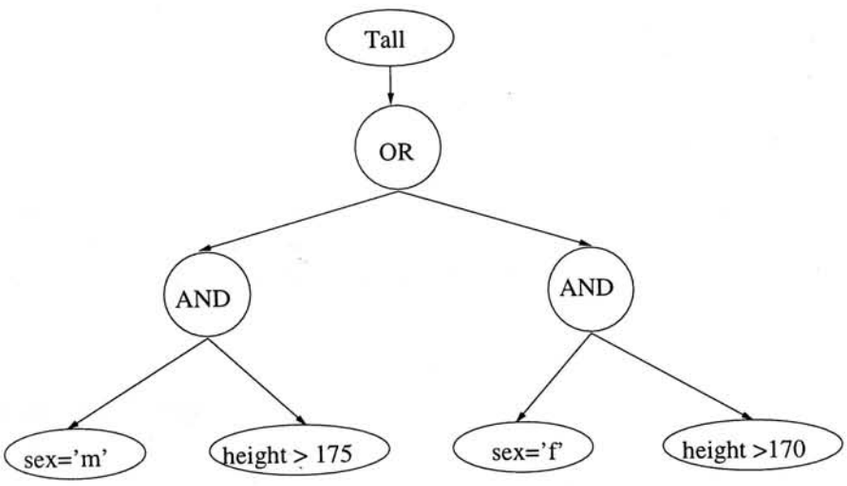


Figure 5.2: The "tall" tree

3. The system will link the concept tree and the temp tree. If the concept still contains subconcepts, the subconcepts will be transformed into more specific subconcepts or conditions. The procedure continues until no concepts are contained in the whole condition tree. In the above example the condition tree is shown in Fig 5.3.

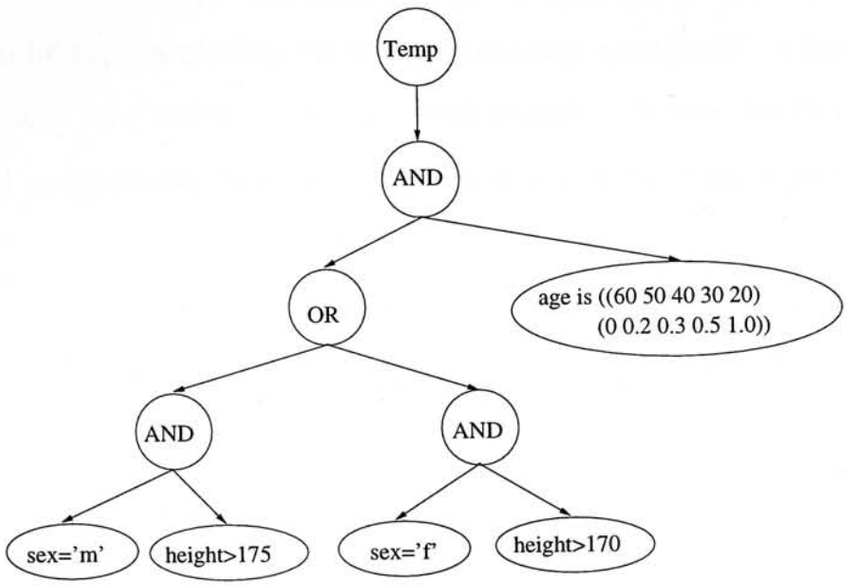


Figure 5.3: The whole condition tree

4. After the whole condition tree has been obtained, the system will convert the whole condition tree into a string of conditions, which are recognized by ORACLE. For the example above, the string of conditions converted from the whole condition is as follows:

"((sex='m' and height > 175) or (sex='f' and height > 170)) and
(age is ((60 50 40 30 20)(0 0.2 0.3 0.5 1.0))) "

5.2.2 Concept trees

A concept tree can be inputed into the Knowledge Base in two ways, namely (1) by loading a concept file; or (2) by interactive input, as mentioned in section 4.4.

We considered two kinds of structure for concept trees when building up our Knowledge Base. One is a complete tree, and the other is a partial tree. A complete tree is one with all its leaves being fuzzy conditions and/or crispy conditions, whereas a partial tree is one whose leaves can be fuzzy conditions, crispy conditions and/or subconcepts.

For example, the term "professional teacher" involves a concept "professional". The concept can be represented by "AND(experienced, qualified)", where "experienced" and "qualified" are subconcepts of "professional". These three concepts can be represented respectively by three complete trees in the knowledge Base, as shown in Fig 5.4.

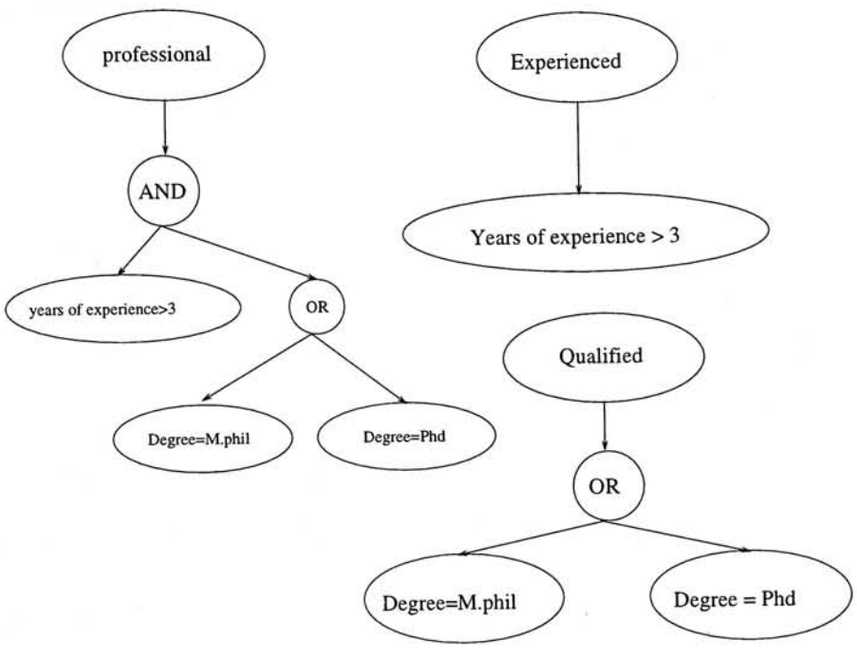


Figure 5.4: Complete trees

It is easy to obtain the concept tree for a concept if in the Knowledge Base each concept is represented by a complete tree. However, this method has the disadvantage that modification of concept trees is very difficult. To maintain the consistency, all the related trees have to be modified at the same time whenever a concept tree is changed. For example, when we modify the concept "experienced", we must modify the concept "professional" at the same time.

The Knowledge Base can be built up with some concepts being represented by partial trees. In this case, the representation of the three concepts in the example above becomes those in Fig 5.5.

Linking partial trees takes a bit longer time than manipulating complete trees.

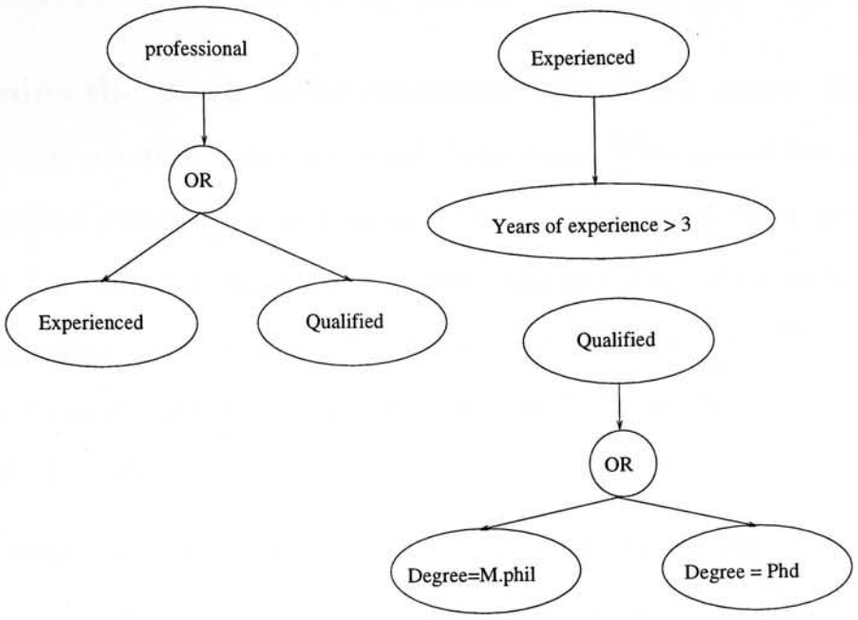


Figure 5.5: Partial trees

However, the modification of partial trees is very easy. If we wanT to modify the concept "experienced", we need to modify the concept "experienced" only.

We chose partial trees to be the structure of concept trees in our Knowledge Base. This data structure is much more convenient to modify, and can avoid data redundance.

5.3 Data Manager

5.3.1 Some issues on the implementation

In our system, there are two types of columns or items, one involving the database fields only and the other involving external functions. The function of the Data Manager is to evaluate the values of external functions. To evaluate an external function, the values of its parameters will be retrieved from the Database first and then put in the `Temp` table. The function value will be computed after knowing the values of the parameters, through

the dynamic library. (The dynamic library will be described in the sequel).

How to determine the items to be retrieved ? In a user query, the item list and conditions may contain some external functions. The parameters of an external function may be items in some relation tables or some other external functions which have items (or external functions) as their parameters. To evaluate the values of external functions, we must get the values for those items. Thus, a problem the system has to deal with is to determine which items are to be retrieved in order to evaluate all the external functions.

We apply some specially designed data structures to generate the items required. `Itemtok` and `Itemname` are two of them, which are given as examples:

The `Itemtok` structure is defined as follows, which is to convert the user's item list expressed in the `string` type to the following type:

```
typedef enum itemtype{
     ITEM_COLNAME, ITEM_FUNC
} ItemType;

typedef struct itemtok{
     ItemType ittype;
     char *colname;
     UserFunc *itfunc;
     struct itemtok *next;
}ItemTok;
```

Note that the names of the items in the `temp` table can be different from the names of the itmes in a user query. Thus, in order to maintain their one-to-one correspondence relationship, we design a data structure `Itemname` as follows:

```
typedef struct itemname {
        char name[NAMELEN];
        char item[ITEMLEN];
        UserFunc *funcptr;
}ItemName ;
```

The steps of determining the required items are below:

1. The Data Manager generates the item list in the `Itemtok` structure. This item list consists of all items and all non-functional parameters of the user's query. This item list will be used in the intermediate SQL (see section 4.7).
2. Meanwhile, the system will generate another item list for the `temp` table. This item list, in the `Itemtok` structure, consists of all items, all parameters and all external functions of the user's query. Those item names, parameters and functions will be automatically re-named, for convenience, by the system with the `Itemname` structure.

How to define the types of the items in the temp table ? The `temp` table contains items, external functions, and degrees of membership, as we have discussed before. The type of the items retrieved from the Database by the intermediate SQL query is defined as `string`. The type of an external function is the same as that of the function value returned from the user dynamic library. The type of a membership degree is `float`.

How to put the records into temp table ? In the temp table, records retrieved by the intermediate SQL query, select_sql_query, are inserted into the temp table by the SQL statement

"INSERT INTO temp (column names) select_sql_query".

This forms an intermediate table which has not obtained the values of external functions yet.

The Data Manager will fetch records in the intermediate table to evaluate the external functions, which will then update the function values in the corresponding columns, using the SQL statement

"UPDATE temp SET func_col_name = func_result WHERE ROWID = fetch_record_row_id".

The membership degree under the fuzzy conditions in the user query will be computed by the Fuzzy processor after the values of the external functions have been obtained. The membership degree is then stored into the degree column of the `temp` table by using the UPDATE statement.

5.3.2 Dynamic library

When the user external functions appear in the user query, our system will use dynamic library to link the external functions. What is Dynamic library?

Function libraries can be divided into two classes, static and dynamic libraries. The static library contains the function codes which are included into the main program file at compile-time. The dynamic library contains functions which are linked to the main program at run-time. It is opposite to the static library which uses static linking mechanism. Static linking binds function name to function address at compile-time. Dynamic linking performs the binding at run-time.

Dynamic linking allows the change of the dynamic library without re-compiling the main program. With dynamic linking, a compiled program can execute with different libraries with a predefined interface. Users can supply their own libraries for a complied program to use.

The set of functions and their prototype should be well defined between the compiled program and the dynamic library. The function prototype includes the function name,

the number of parameters, parameter types, result types, etc.

Our system needs to manipulate external functions whose prototypes are not defined before compilation. This requires features more dynamic. Thus, we have to enhance the dynamic library, which is described in the subsequent sections.

5.3.3 Precompiling process

External functions will be created to perform general purpose computing and database access. The external function can be written in C programming language or embedded SQL C language. We put the C function in .fc file. For example, function `reverse()` returns a string in the reverse order of the parameters, and function `substr()` returns a substring of the parameter. We put the embedded or dynamic SQL C function in .fpc file, e.g., function `avgage()` returns the age average. By dynamic SQL, function `avg(tablename,colname)` returns the average of parameter `colname` in the `tablename`.

How are the external functions compiled and how can the system find the external functions in the dynamic library? The steps are as follows:

1. External functions are written in a file (.fpc or .fc).

2. A precompiler is constructed in order to generate an interface between the external functions and the system.

3. This file (.fpc or .fc) will be precompiled by the precompiler which we have defined already and this file will be changed to .c file and an interface will be generated which contains the external function's information.

4. When the system calls an external function from dynamic library, it will call this interface in order to get the external function information. The interface has two functions. They are `usrfunctype()` and `usrfunc()` whose type is FuncResStruct. The first one defines the function type, in which we use `strcmp` statement to match the function name to return the function type. The second one is to malloc a result pointer and pass the function name and parameters to the extfunc[ ]. In this

function, we use string compare statement `strcmp` to match the function name so as to generate the output.

5. The .c file will be compiled by the C compiler. The .c file will be changed to .o file.

6. The .o file will be linked to the C library or the ORACLE library and then changed to .so file.

7. The .so file will be connected to the dynamic library. Then, the system will know how to execute the external function and pass the parameters to get the required value.

The flow chart of the steps is shown in Fig 5.6.

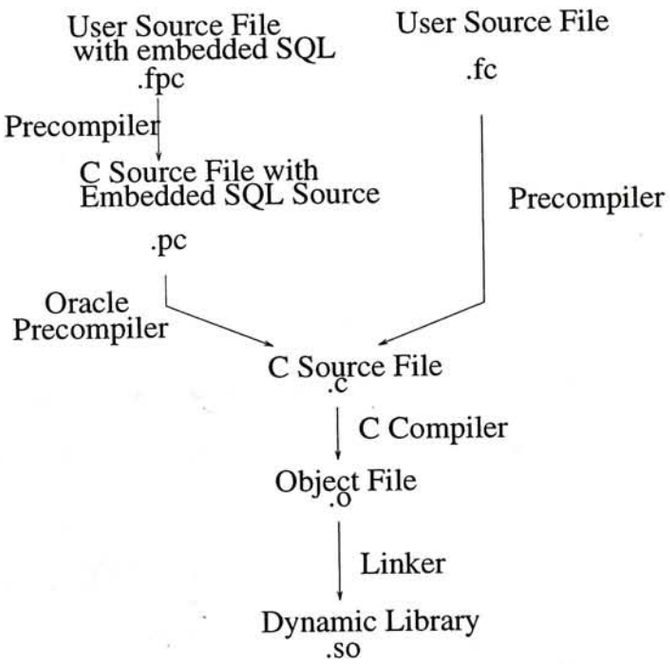


Figure 5.6: The precompiling process

For example, Suppose that an external function `substr()` is designed as follows:

```
char *substr(s,pos,len)
char *s ;
```

```
int pos, len ;
{
     char *result ;

     result = malloc(len+1);
     strncpy(result,s+pos,len);
     result[len] = '\0' ;

     return result ;
}
```

The external function `substr()` will be precompiled and an interface file will be generated as follows.

```
FuncResType usrfunctype(funname)
char funname[] ;
{
     if ( !strcmp(funname,"substr") || !strcmp(funname,"SUBSTR") )
          return FR_STRING ;
     else
          return FR_VOID ;
}

FuncResStruct *usrfunc(extfunc)
char *extfunc[];
{
     FuncResStruct *result ;
```

```
        result = (FuncResStruct*) malloc(sizeof(FuncResStruct));
        if ( (!strcmp(extfunc[0],"substr") || !strcmp(extfunc[0],"SUBSTR"))
             && extfunc[1] != NULL && extfunc[2] != NULL
             && extfunc[3] != NULL && extfunc[4] == NULL )
        {
             char *p1 ;
             int p2 ;
             int p3 ;
             p1 = extfunc[1] ;
             p2 = atoi(extfunc[2]) ;
             p3 = atoi(extfunc[3]) ;

             result->frtype = FR_STRING ;
             result->value = substr(p1,p2,p3) ;
        }
        else
        {
             result->frtype = FR_VOID ;
        }
        return result ;
    }
```

The precompiler above generates the output according to the calling standard which is described in the following sub-section.

5.3.4 Calling standard

How is the information on a function passed to our system ? The main information on a function includes function name, function type, number of parameters, parameters and

each parameter type. As we described above, an interface between the system and external functions is designed. Through the interface, the information of an external function can be passed to the system and the system can invoke the external function.

The process of calling functions from the dynamic library is illustrated in Fig 5.7.

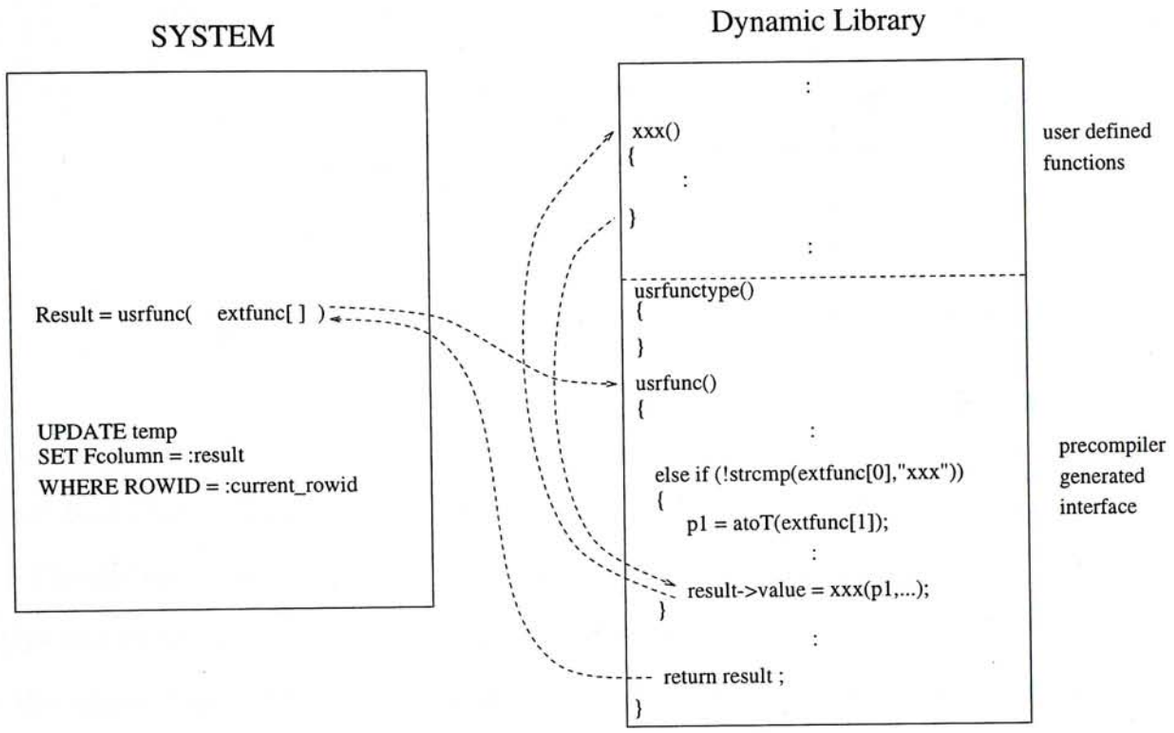


Figure 5.7: The calling process

5.3.4.1 Function Invocation

When we call external functions, the precompiler will pass the function name and parameters to variable `extfunc[]` automatically. The type of the variable `extfunc[]` is an array of character pointers. Each of them will point to function name or parameters. The structure of external function invocation is shown in Fig 5.8.

Let us consider an example of `substr("abcdef",1,3)`. The name of function is `substr`. The function type is character pointer. The first parameter is a string `"abcdef"`. The second parameter is an integer `"1"` and the last parameter is an integer `"3"`. So the

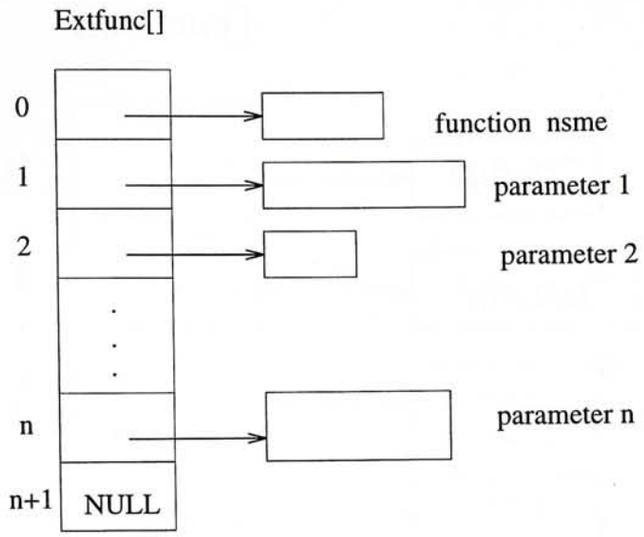


Figure 5.8: The structure of external function invocation

structure of extfunc[] is shown in Fig 5.9.

How does the system get the function value after it passes the information of the function ? Firstly, we compare the function name with `extfunc[0]`. If they match, we will process the `extfunc[n]`(n>0) in order to let the function parameter and `extfunc[n]`(n>0) have the same type. The type of `extfunc[]` is string. So when we pass the parameter to the interface, we must convert string type to user defined parameter type. Usually we convert the string type to integer type and float type using the library function `atoi()` and `atof()`. The processing of function invocation is done by the precompiler. e.g. the parameters of `substr()` are `extfunc[1]`, `extfunc[2]` and `extfunc[3]`. The type of `extfunc[]` is character pointer. The type of function parameter `"1"` and `"3"` are integer. So we will change the character pointer to integer using C library function `atoi()`. Then we pass the value of `extfunc[n]`(n>0) to the function parameter. The value of the function is equal to `substr("abcdef",1,3)`. The type of the function is string. All these information will be returned to the system.

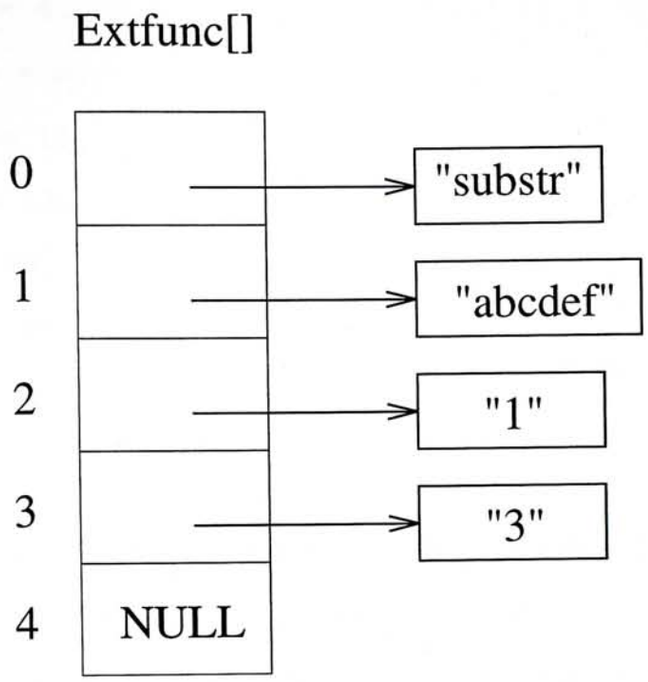


Figure 5.9: The structure of extfunc[]

5.3.4.2 Function Result

We define a data structure named FuncResStruct as follows, to represent the function type and the result:

```
typedef struct funcresstruct {
    FuncResType frtype ;
    void *value ;
} FuncResStruct ;
```

There are different function types, `frtype`, which are defined as follows:

```
typedef enum funcrestype {
    FR_VOID = 0,
    FR_VARCHAR = 1,
    FR_INT = 3,
```

```
        FR_FLOAT = 4,
        FR_STRING = 5,
        FR_ROWID = 11,
        FR_DATE = 12,
        FR_RAW = 23,
        FR_CHAR = 96
    } FuncResType ;
```

The codes above are the external datatype codes for ORACLE dynamic SQL. This allows the function result to be stored into the column of our *temp* table.

The field `value` points to the value whose type is represented on the field `frtype`. So the type of the value is dynamic. If the type of function A is INT and the value is 40, the structure is depicted in the Fig 5.10.

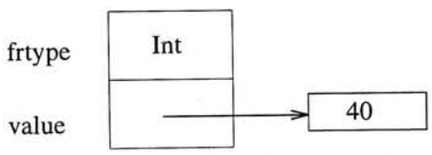


Figure 5.10: The structure of function A

Chapter 6

CASE STUDIES

A job center performs as an agent for applicants as well as for employers. On one hand, they help applicants to find jobs they are looking for. On the other hand, they try to allocate suitable candidates to vacancies to satisfy the requirements of employers. To better perform these duties, they usually maintain a database to keep data on applicants and data on employers. These data usually are the information on applicants, such as their personal details, qualifications, experience, skills, preferences, etc, and the information on employers, such as their vacancies, job description, requirements, remunerations, etc. When an applicant is looking for a job, they may use the database to identify the possible vacancies that best satisfy his preference . Similarly, An employer may use the database to find those potential candidates who best meet the job requirements. In both cases, the database queries may involve a number of fuzzy conditions and/or concepts, like a *good* candidate *preferably* with a degree in *computer related areas*, a job which is not *too far* away from my home, a job with which I may get the *highest possible* reward, etc. In these situations, our fuzzy database query system would be a very good tool to solve the problems. Applications of the query system to these situations would be good examples to test the effectiveness of the system. For this purpose, we have developed a sample database on human resource management. In the following we shall apply our system to some cases to test the capabilities of the system.

6.1 A Database for Job Application/Recruitment

An experimental database (see Appendix A) for job application/recruitment has been developed, which contains eleven tables, which are **company**, **position**, **salary**, **required qualification**, **required language**, **speciality**, **applicant**, **applicant qualification**, **school**, **language**, and **job history**. The logical relationship between the entities related to an open position is shown in the E-R diagram of the fig 6.1 and the logical relationship between the entities related to an applicant is shown in the E-R diagram of the fig 6.2. Each table is briefly introduced below:

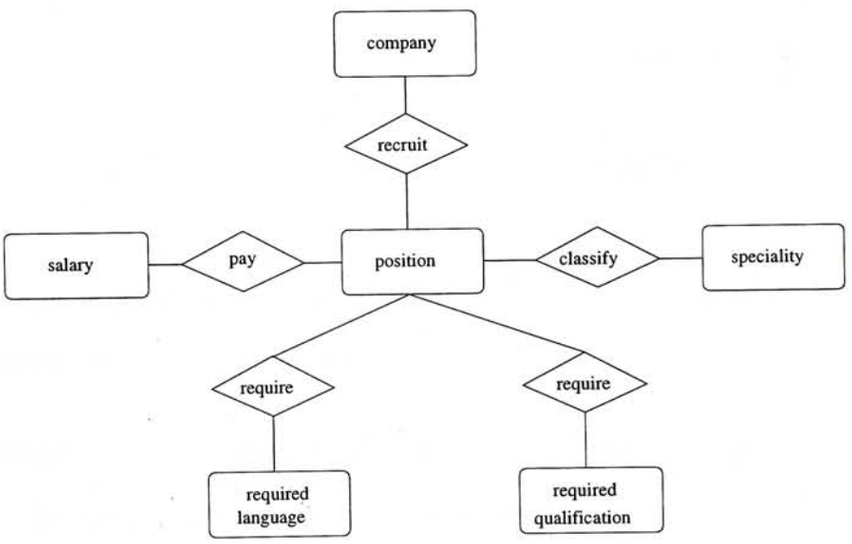


Figure 6.1: The E-R diagram for **position**

company = (company-id, company-name, business, city, state, contact-phone-number) contains the companies which have vacancies,

position = (company-id, position-name, sex-of-candidate, age-limits) contains information on the open positions,

salary = (company-id, position-name, years-of-experience, corresponding-salary) contains information on the salary of the positions,

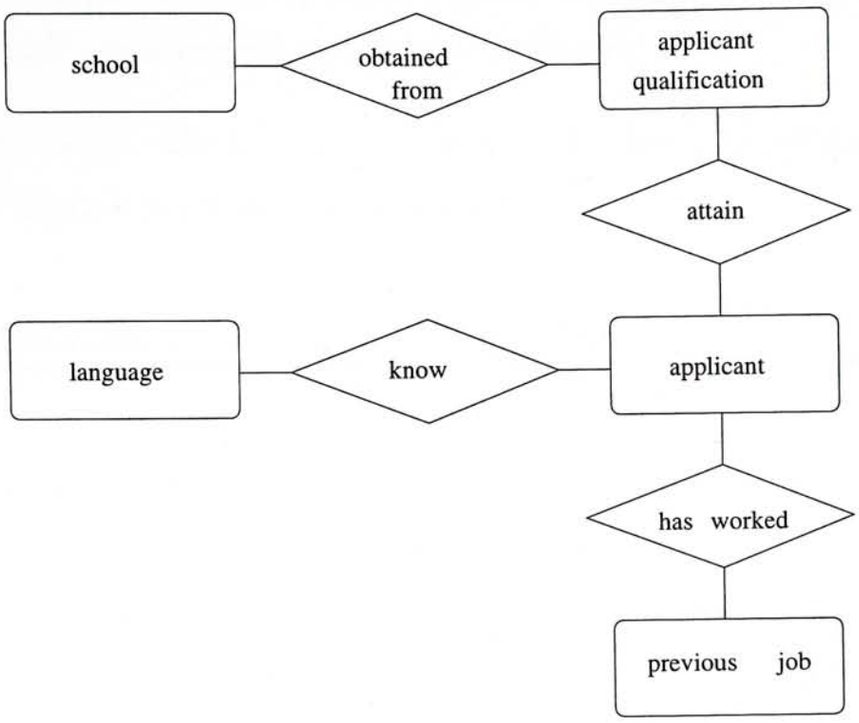


Figure 6.2: The E-R diagram for **applicant**

speciality = (company-id, position-name, major, minor) contains information on the required speciality of the positions,

required language = (company-id, position-name, required-language) contains information on the language required for the positions,

required qualification = (company-id, position-name, certificate) contains information on the qualification required for the positions,

applicant = (applicant-id, applicant-name, sex, age, address, city, state, height, weight, marital-status) contains information on the applications,

applicant qualification = (applicant-id, certificate, school, study-subject, grade, graduatic year) contains information on the qualification of the applicants,

language = (applicant-id, languages) contains information on the language skills of the applicants,

job history = (applicant-id, company-id, position-name, date, salary) contains information on the employment history of the applicants,

school = (school-code, school-name, city, state) contains information on the schools where the applicants got their qualifications.

6.2 Introduction to the Knowledge Base

The Knowledge Base of the system has to contain some domain knowledge about the concepts involved in problems of job application and recruitment. For the purpose of testing, we have included some related concept trees in an experimental knowledge base we have built in our system, which is shown in fig 6.3.

Remark sumlookup1('col_to_sum','ref_table','col_to_lookup',match_val) is predefined in our system. This function looks up the table ref_table for the records with the col_to_lookup field being value of match_val. It returns the sum of the col_to_sum field from all satisfied records.

6.3 Cases

In the following we will look at some illustrative examples to see how our system works, and whether it can achieve its expected capabilities. We will test several cases which involve respectively crispy queries, fuzzy queries, concept queries, fuzzy operators, etc.

6.3.1 Crispy queries

A Solicitor & Co, Ltd. has some vacancies. This company wants to fill the positions by some college graduates, under the conditions that they should be male, between 20 and

35, and majoring in law. The company may thus wish to find the names, certificates and GPAs of the most suitable candidates under these conditions.

This query is a simple one, since it contains some crispy conditions only. According to the syntax of our query language, the query is as follows:

```
SELECT applic_name, certificate, gpa
FROM applicant x, app_qualifi y
WHERE and (x.applic_id = y.applic_id, sex = 'M',
     major = 'Law', age >= 20, age <= 35 )
```

In our database, the `applicant` table contains the information on the applicants (e.g. applicant name, sex, ...), and the `app_qualifi` table contains the information on the qualifications of applicants (e.g. the certificate, major, etc.). Thus, the two tables "applicant" and "app_qualifi" should be joined to get the results required by the query. After the query was submitted, our system returned the results from the database, as shown in the table 6.1.

Table 6.1: Crispy Query Result 1

APPLIC_NAME	CERTIFICATE	GPA	DEGREE
Rick Canonica	Bachelor	2.08	1.000
Rick Clairmont	Bachelor	2.29	1.000
Daniel Emery	Bachelor	2.4	1.000
Norman Nash	Bachelor	2.85	1.000
Leslie Andriola	Bachelor	3.41	1.000
Bill Johnson	Bachelor	3.81	1.000
Bill Johnson	Doctor	3.79	1.000
Rick Clairmont	Master	2.88	1.000
Larry Dement	Master	3.54	1.000
Bill Johnson	Master	3.7	1.000

The **DEGREE** in the Table above stands for the membership degree of an alternative in terms of satisfying all the query conditions. In the present case, the degrees of all the

alternatives are 1.0 because the conditions are crispy and all the alternatives retrieved from the database exactly satisfy these conditions.

We now consider the following example which involves the use of external functions. Suppose that when Solicitor & Co, Ltd. considers the applicants, it imposes one more condition, which requires that any applicant should have more than 8 years of working experience. Besides, the company is interested to know the address of the applicants as it wants to find those who live near to the company. In this example, we used two external functions in the query. One is the `concat()`, which concatenates its string parameters `address` and `city`. The other is the `sumlookup1()` function, which looks up the applicant's job history and sums up the working years. The query required is as follows:

```
SELECT applic_name, certificate, gpa, concat(address,city)
FROM applicant x, app_qualifi y
WHERE and (x.applic_id = y.applic_id, sex = 'M',
     major = 'Law', age >= 20, age <= 35,
     sumlookup1('(end_date-start_date)/365',
     'jobhistory', 'applic_id', x.applic_id) > 8 )
```

After this query was submitted, our system returned as in the table 6.2.

Table 6.2: Crispy Query Result 2

APPLIC_NAME	CONCAT(ADDRESS,CITY)	CERTIFICATE	GPA	DEGREE
Rick Clairmont	92 Madiso7 Drive Chocorua	Bachelor	2.29	1.000
Norman Nash	87 West Rd. Meadows	Bachelor	2.85	1.000
Rick Clairmont	92 Madiso7 Drive Chocorua	Master	2.88	1.000
Larry Dement	166 Williams Rd. Etna	Master	3.54	1.000

From the above two examples we see that our system is able to handle crispy conditions very easily. The last example shows that the system successfully used external functions

in a query. An external function can appear either in the item list or in the condition list of a query.

6.3.2 Fuzzy queries

In many cases, users want to query with imprecise conditions. Our system can handle not only crispy conditions but also fuzzy conditions. In [42], the membership function for a fuzzy condition is represented by two lists of data. In our system, the membership function for a fuzzy condition can be represented either by two lists of data or by an external function. This provides a very general and flexible way to do the representation, since a user can choose any function he deems appropriate to express his membership function. In the following we will show this capacity of the system by using some examples.

6.3.2.1 Fuzzy conditions expressed by two lists of data

In this example, the fuzzy condition in a query is presented in the following format:

COLUMN_NAME is ((VALUE_LIST)(DEGREE_LIST))

Suppose that a Newspaper Office is recruiting photographers under the conditions that: ages are between 20 and 35 inclusively, tall, major in Arts, can speak English.

The query is as follows:

```
SELECT applic_name, age, height
FROM applicant x, app_language y, app_qualifi z
WHERE and( x.applic_id = y.applic_id, x.applic_id = z.applic_id,
     age >= 20, age <= 35, field = 'Arts', lan_name = 'English',
     height is ( (155 160 165 170 175 180 185 190)
          (0.0 0.0 0.1 0.3 0.5 0.8 0.9 1.0) ) )
```

In this query, the fuzzy condition `tall` is represented by two lists. The first list are the heights values, while the second list are their corresponding degrees of membership

in the fuzzy set *tall*. We submitted this query to our system. The PARSER module then processed it and generated a SQL query as follows:

```
SELECT applic_name, age, height
FROM applicant x, app_language y, app_qualifi z
WHERE x.applic_id = y.applic_id and x.applic_id = z.applic_id
      and age >= 20 and age <= 35
      and field = 'Arts'and lan_name = 'English'
      and height > 160
```

After processing, the following alternatives together with their degrees of membership to satisfy all the conditions were obtained as in the table 6.3

Table 6.3: Fuzzy Query Result 1

APPLIC_NAME	AGE	HEIGHT	DEGREE
Paul Belliveau	30	185	0.900
Christine Ulrich	35	177	0.620
Johanna Gaudet	20	171	0.340

6.3.2.2 External function in the two-list expression

In the following example, we will consider the case where an external function appears in the fuzzy condition which is expressed by two lists of data. The format is:

USERFUNCTION is ((VALUE_LIST)(DEGREE_LIST))

Suppose that, in addition to the criteria as given in section 6.3.2.1, the Newspaper Office adds one more condition, namely, a qualified applicant should have *about eight years* of working experience. The final results should include the columns in section 6.3.2.1 and the years of working experience. The query is shown below, in which an external function is used in the fuzzy condition *about eight years* of working experience.

```
SELECT applic_name, age, height,
     sumlookup1('(end_date-start_date)/365',
     'jobhistory', 'applic_id', applic_id)
FROM applicant x, app_language y, app_qualifi z
WHERE and( x.applic_id = y.applic_id, x.applic_id = z.applic_id,
     age >= 20, age <= 35, field = 'Arts', lan_name = 'English',
     height is ( (155 160 165 170 175 180 185 190)
          (0.0 0.0 0.1 0.3 0.5 0.8 0.9 1.0) ),
     sumlookup1('(end_date-start_date)/365', 'jobhistory',
          'applic_id', applic_id) is
          ((0 2 4 6 8 10 12 14 16)(0.1 0.2 0.45 0.85 1 0.85 0.45 0.2 0.1
```

In the query, the function sumlookup1() retrieves the years of working experience, and the fuzzy set *about eight years* is represented by two lists of data. After submitting this query, our system returned the following results, in which the alternatives are sorted by descending order of the values in the column DEGREE, which is the degree of membership that an alternative satisfies all the conditions, in table 6.4.

Table 6.4: Fuzzy Query Result 2

APPLIC_NAME	AGE	HEIGHT	SUMLOOKUP1()	DEGREE
Paul Belliveau	30	185	7.821918010711	0.800
Christine Ulrich	35	177	14.750684738159	0.162
Johanna Gaudet	20	171	0	0.100

6.3.2.3 Membership expression by external function

We now examine the problem of expressing degrees of membership directly by using an external function. The format is:

USERFUNCTION

This external function will return a membership degree directly when its parameters are given. We consider the example mentioned before. But this time the degrees of membership for the fuzzy condition *about eight years* is represented by an external function. The query is rewritten as follows:

```
SELECT applic_name, age, height,
     sumlookup1('(end_date-start_date)/365',
     'jobhistory', 'applic_id', applic_id)
FROM applicant x, app_language y, app_qualifi z
WHERE and( x.applic_id = y.applic_id, x.applic_id = z.applic_id,
     age >= 20, age <= 35, field = 'Arts', lan_name = 'English',
     height is ( (155 160 165 170 175 180 185 190)
          (0.0 0.0 0.1 0.3 0.5 0.8 0.9 1.0) ),
     normdist( 0, 16, sumlookup1('(end_date-start_date)/365',
     'jobhistory', 'applic_id', applic_id) ) )
```

Remark

`normdist ( minimum_value, maximum_value, value )` is predefined in our system, which is a normal distribution. The parameters `minimum_value` and `maximum_value` are the range of the distribution. The membership degrees for the points at and beyond the end points of the range of the distribution are set to be 0. The degree at the mean of the distribution is set to be 1. The normdist() function returns the membership degree at the parameter `value`.

The final results obtained by our system with the query are in table 6.5.

6.3.3 Concept queries

Suppose that the Computer Center of a HK University is recruiting a CO position. The query is to find a *suitable candidate* as a Computer Officer, where *suitable candidate* is a

Table 6.5: Fuzzy Query Result 3

APPLIC_NAME	AGE	HEIGHT	SUMLOOKUP1()	DEGREE
Paul Belliveau	30	185	7.821918010711	0.800
Christine Ulrich	35	177	14.750684738159	0.058

concept, which has been predefined in our knowledge base (see section 6.2). So the query is generated as follows:

```
SELECT applic_name, certificate, age
FROM applicant, app_qualifi
WHERE and(applicant.applic_id = app_qualifi.applic_id,
     major='Computer', suitable_candidate,
```

we submitted the above query to our query system. The query was firstly processed by the knowledge base, since it contained a concept. The knowledge base transformed the concept into some crispy and fuzzy conditions, and then sent them to the Parser, where they were combined with the original conditions to generate the following query:

```
SELECT applic_name, certificate, age
FROM applicant, app_qualifi
WHERE and(major='Computer',
     applicant.applic_id = app_qualifi.applic_id,
     and (age is ((45 40 35 30 25 20 15) (0.0 0.2 0.5 0.8 1.0 0.5 0.0)),
     or ( certificate = 'Master', certificate = 'Doctor'),
     or ( and ( certificate = 'Master', sumlookup1(...) > 7),
     and ( certificate = 'Doctor', sumlookup1(...) > 5))) )
```

Based on the query above, the Parser further generated the following SQL query containing crispy conditions only:

```
SELECT applic_name, certificate, age
FROM applicant, app_qualifi
WHERE major='Computer' and
      applicant.applic_id = app_qualifi.applic_id,
      and age > 15 and age < 45 and
      (certificate = 'Master' or certificate = 'Doctor') and
      (certificate = 'Master' or certificate = 'Doctor')
```

Those records which were retrieved from the Database under the above SQL query were processed by the Fuzzy Processor. The Fuzzy Processor determined the degrees of membership that the candidates satisfied the fuzzy conditions, and returned those with non-zero degrees of membership to the user.

The query system was able to successfully handle the concept query, and finally returned the records from the the Database, as in table 6.6.

Table 6.6: Concept Query Result

APPLIC_NAME	CERTIFICATE	AGE	DEGREE
Christine Villari	Doctor	38	0.320
Christine Villari	Master	38	0.320
Meg Dallas	Master	40	0.200

6.3.4 Fuzzy Match

We now consider an example which involves a condition about to what degree a subject matches with another subject. Suppose that a TV Broadcasting Company, with id 00138, wants to find the qualified candidates for its available vacancies. For each vacancy there is a requirement on the major of the candidate. However, an applicant with a major in a related area is also considered acceptable. For example, for the vacancy `Electronics Engineer`, it is hoped that a candidate should preferably major in electronics engineering, but some one majoring in computer science may also be acceptable. The required

information is the name, certificate, gpa of each candidate together with the vacancy that he is qualified for. The query in this case is as follows:

```
SELECT po_name, applic_name, certificate, gpa
FROM applicant x, app_qualifi y, speciality z
WHERE and ( z.co_id = '00138', x.applic_id = y.applic_id ,
          dblookup2('fieldmap', 'major1', z.major, 'major2', y.major) )
```

The conditions "x.applic_id = y.applic_id" and "z.co_id = '00138' are crispy conditions. The table "applicant' and "app_qualifi' are joined together. The condition "dblookup2()" is a fuzzy condition. The dblookup2() will return membership degrees when the field 'major1' and 'major2' are matched with the values major1 and values major2. It is an external function predefined by the system.

The final result obtained by our query system is in the table 6.7.

Remark

dblookup2('ref_table','col1',val1,'col2',val2) is predefined in our system. This function looks up the table ref_table by the fields col1 and col2. The fields col1 and col2 are matched with the values val1 and val2. The function returns the field of membership degree of the matched record. The function acts as a membership degree function through table lookup.

6.3.5 Fuzzy operator

We now consider a situation to demonstrate the application of our fuzzy operators. Suppose that a company, Leung & Wong and Associates, Ltd., is looking for a person to fill an important tenurable position. The company has a set of criteria, like the person must hold a master degree, major in computer science or a related area, and have suitable experience. As the person is most likely to be tenured after a short period of probation, the company wishes to fill up the position carefully. It is normal that, at the stage of

Table 6.7: Fuzzy Match Result

PO_NAME	APPLIC_NAME	CERTIFICATE	GPA	DEGREE
Electronics Engineer	Norman Lasch	Bachelor	3.07	1.000
News Editor	Rick Foote	Bachelor	3.1	1.000
News Editor	Kevin Manning	Doctor	3.19	1.000
News Editor	Kevin Manning	Master	3.78	1.000
Electronics Engineer	Kathleen Robinson	Bachelor	3.01	1.000
News Editor	Len Mercier	Doctor	3.39	1.000
News Editor	Len Mercier	Bachelor	3.08	1.000
Electronics Engineer	Karen Clarke	Bachelor	2.2	1.000
News Editor	Alvin Toliver	Bachelor	2.48	1.000
News Editor	Chris Danzig	Bachelor	3.28	1.000
.....	.....	.....	.....	....
News Editor	Janis Gehr	Bachelor	3.04	0.700
News Editor	Mary McElroy	Bachelor	3.17	0.700
News Editor	Mary McElroy	Master	3.61	0.700
Electronics Engineer	Meg Dallas	Master	3.65	0.700
News Editor	Susan Gray	Bachelor	1.58	0.700
News Editor	Susan Gray	Master	3.41	0.700
.....	.....	.....	.....	....
.....	.....	.....	.....	....

recruitment, a candidate always tends to demonstrate his capabilities to meet all the conditions perfectly. However, after he has secured the job, one may find that he in fact does not match one or more of the required criteria. In that case, the decision maker would feel a kind of regret (on hiring this person). The objective of the present company is to minimize the degree of regret that may occur.

This is an example in which we should apply our operator REGRET. The query for this example is as follows:

```
SELECT applic_name, age
FROM applicant, app_qualifi, speciality
WHERE and( applicant.applic_id = app_qualifi.applic_id,
     speciality.co_id = '00138', certificate = 'Master',
     regret(dblookup2('fieldmap','major1',speciality.major,'major2',
     app_qualifi.major),
     normdist(0,20,sumlookup19'(end_date-start_date)/365',
     'jobhistory','applic_id',applicant.applic_id) )
```

The final results obtained by our system are as in table 6.8.

Table 6.8: Result when FUZZY_OPERATOR is "REGRET"

APPLIC_NAME	AGE	REGRET DEGREE	COMPLEMENT REGRET DEGREE
Louie Ames	36	0.011	0.989
Rick O'Sullivan	37	0.060	0.940
Rick Dietrich	32	0.147	0.853
Mary Lou Jackson	30	0.338	0.662
Mary Lou Morrison	38	0.465	0.535
Susan Gray	28	0.499	0.501
Peter Lengyel	42	0.604	0.396
Lisa Rodrigo	42	0.620	0.380
Kathleen Clinton	41	0.625	0.375
George Siciliano	27	0.662	0.338
Mary Clarke	43	0.690	0.310
Marjorie Clinton	26	0.764	0.236
Jo Ann Lapointe	31	0.767	0.233
Dean Mellace	41	0.798	0.202
Alvin Dement	39	0.800	0.200
Louis Reynolds	40	0.800	0.200
Larry Nunez	34	0.800	0.200
Hope Stadecker	34	0.800	0.200
Peter Blount	26	0.820	0.180
Roger Saninocencio	41	0.823	0.177
Janis Peters	25	0.889	0.111
Ann Boyd	25	0.934	0.066
Peter Flynn	25	0.935	0.065
Mary McElroy	23	0.936	0.064

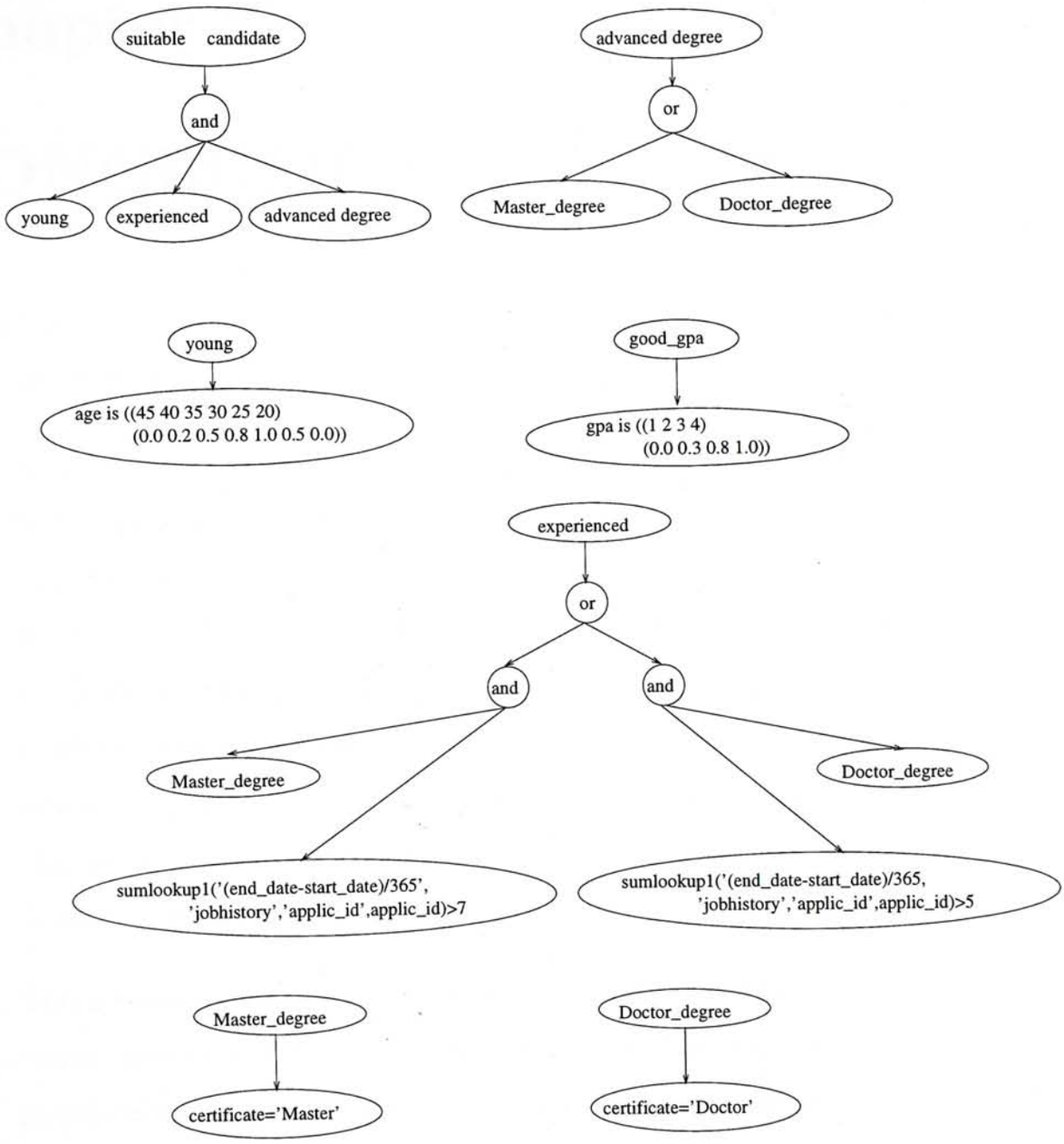


Figure 6.3: The knowledge Base in case study

Chapter 7

CONCLUSION

A fuzzy query system has been developed, which has been designed to possess the following main capabilities:

1. A Knowledge Base is built in the system, which can deal with queries involving highly-aggregated, imprecisely and vaguely-defined concepts by further decomposing them into more specific and well-defined conditions. In the knowledge base, each concept is represented by a hierarchical concept tree. Each node of a tree may be a sub-concept, an operator, or a fuzzy or crispy condition. In addition to providing a natural structure to represent general concepts, other advantages of concept trees include easy maintenance, easy modification, and avoidance of data redundancy. The use of a knowledge base consisting of concepts trees to handle general concepts is one of the main contributions of this thesis.

2. The system supports the use of external functions, which can be utilized to represent membership functions. As we reviewed before, how to effectively express membership functions, particularly when more than one attribute are involved, is one of the main issues in dealing with imprecise queries using the fuzzy sets theory. The external functions supported by our system can be general functions of multiple variables, which provides a very general approach to represent the multidimensional membership functions, which seems to be more flexible and powerful as compared to previous approaches.

3. The system supports six basic fuzzy operators, on which we have provided a new explanation from the perspective of decision making under uncertainties. In addition to the usual functions of combining fuzzy conditions in the normal sense (such as AND, OR, etc.), our approach should be particularly useful when a user is facing a situation where he wants to search for some optimal solutions from the database against some uncertain futures.

4. The system can be added directly on to an existing database management system to make it capable of accepting and handling imprecise queries, without imposing any more requirement to further change and modify the existing system. As we stated earlier, this is one of our main objectives.

The fuzzy query system has been tested and evaluated on an experimental database. Some cases have been tested, which suggests that the system possesses the expected capabilities to accept and handle user queries involving highly-aggregated concepts, fuzzy conditions as well as ordinary crispy conditions.

Some topics for future research are as follows:

1. An interesting future work is to build up certain mechanism in the knowledge base so that it can learn automatically. For example, in the present system, a different user may provide a different definition about a fuzzy condition. This in fact contains certain knowledge from the user about the fuzzy condition/concept. A gradual accumulation of the knowledge from different users may help the knowledge base modify the existing knowledge and add new knowledge.

2. It should be pointed out that the main concern of this thesis is to develop and test our approaches to process fuzzy queries. We have made use of SQL. However, we have used only part of the basic functions of SQL. A future work should be to gradually introduce all functions of SQL into the system.

3. We have applied six operators in the Fuzzy Processors of the present system. There are many other operators that have been proposed in the contexts of the fuzzy sets

theory and the optimal decision making theory. Undoubtedly, a future work of interest is to gradually introduce these operators into the system to further enhance its functions.

4. At present we assume that the database is a conventional one with no fuzziness. An interesting work in the future would be exploring the possibility of applying the fuzzy query system to a database which also contains fuzzy terms.

Appendix A

Sample Data in DATABASE

Sample data of the applicant table are

```
APPL- APPLIC_NAME              S        AGE ADDRESS
ICANT
 ID
----- -------------------- - ---------- ------------------------
CITY              ST     HEIGHT     WEIGHT BIRTHDAY  S
--------------- -- ---------- ---------- --------- -

00232 Mary McElroy             F          23 108 Rocky Pond Rd.
Cambridge         MA         176         121 14-JUL-72 N

00233 Susan Mathias            F          38 24 Oxford St.
Milan             NH         149          87 23-JUN-57 Y

00234 Kathleen Robinson        F          30 160 Copps Hill Rd.
Keene             NH         179          84 21-OCT-65 N

00235 Kathleen Clinton         F          41 52 Meeting House Rd.
Penacook          NH         160         102 10-OCT-54 Y

00236 Ruth Gramby              F          40 146 Hasselbrook Rd.
Etna              NH         156         140 21-FEB-55 N

00237 Frederick Burton         M          31 167 Meeting House Rd.
Penacook          NH         149         110 16-NOV-64 N

00200 Al Ziemke                M          30 121 Putnam Hill Rd.
Winnisquam        NH         169         114 27-OCT-65 N
```

```
00201 Marjorie Clinton      F        26 111 Watersedge Dr.
Wonalancet       NH         175      124 02-OCT-69 N

00202 Margaret Harrington   F        21 183 Freedom St.
Fremont          NH         188      135 05-JAN-74 Y

00203 Johanna Gaudet        F        20 40 Mason Rd.
Fremont          NH         171      158 29-DEC-74 N

00204 Charles Myotte        M        20 78 Lantern Lane
Fremont          NH         154      148 29-JUL-75 Y

00205 Wes Bartlett          M        43 28 Page Hill Rd.
Meadows          NH         160      110 16-APR-52 Y

00206 Marty Stornelli       M        33 25 Searles Rd.
Jefferson        NH         164      140 27-MAY-62 Y

00207 Joseph Babbin         M        20 124 Woodfield St
Sanbornville     CT         152       83 12-DEC-74 N

00208 Joe Sciacca           M        42 200 Sullivan Terrace
Munsonville      NH         158      144 15-FEB-53 N

00209 Roger Smith           M        22 163 Lowell Rd.
Bristol          NH         157      136 17-MAY-73 N

00210 Alan Dietrich         M        31 128 Granite St.
Whitefield       NH         170       97 03-JUN-64 Y

00211 Ernest Gutierrez      M        44 165 Old New Boston Rd.
Farmington       MA         183      163 06-JUL-51 N

00212 Roy Clarke            M        25 187 Underhill St.
Sandown          NH         169      134 06-MAY-70 N

00213 Len Mercier           M        33 67 Oxford St.
Milan            NH         174      106 12-FEB-62 Y
....             ..         ..       ..............
....             ..         ..       ..............
-----  ----------------  ---   ----------------------------
```

Sample data of the applicant qualification table are :

APPL_ IANT ID	CERTIFI_ CATE	SCH_ OOL	FIELD	GPA	CERT_YEAR
00219	Bachelor	PRDU	Law	2.96	1979
00220	Bachelor	MIT	Business Admin	3.04	1992
00221	Doctor	MIT	Prod. Engrg.	3.9	1980
00221	Master	AU	Arts	2.96	1977
00221	Bachelor	STAN	Arts	2.92	1975
00222	Bachelor	STAN	Elect. Engrg.	3.07	1990
00223	Bachelor	BATE	Jouralism	3.1	1979
00224	Doctor	DREW	Jouralism	3.19	1989
00224	Master	DREW	Jouralism	3.78	1986
00224	Bachelor	COLB	Jouralism	3.88	1984
00225	Master	HVDU	Civil Engrg.	2.98	1988
00225	Bachelor	CALT	Civil Engrg.	3.54	1986
00226	Bachelor	CALT	Arts	2.53	1992
00227	Bachelor	BOWD	Translation	3.07	1990
00228	Doctor	FLU	Arts	3.36	1978
00228	Master	FLU	Arts	3.9	1975
00228	Bachelor	STAN	Arts	2.91	1973
00229	Master	YALE	Chemical Engrg.	3.84	1981

```
00229 Bachelor COLB Mech. Engrg.              2.63        1979

00230 Master   PRDU Translation               3.27        1993

00230 Bachelor FLU  Translation               3.18        1991

00231 Master   STAN Law                       2.88        1986

00231 Bachelor BOWD Law                       2.29        1984

00232 Master   DREW Business Admin            3.61        1994

00232 Bachelor UME  Business Admin            3.17        1992

00233 Doctor   BATE Prod. Engrg.              2.81        1983

00233 Master   FLU  Prod. Engrg.               1.7        1980

00233 Bachelor YALE Prod. Engrg.              3.19        1978

00234 Bachelor STAN Elect. Engrg.             3.01        1985

00235 Doctor   HVDU Civil Engrg.              3.19        1980

00235 Master   DREW Civil Engrg.              2.77        1977

00235 Bachelor DREW Arts                       2.6        1975

00213 Master   UME  Jouralism                 1.69        1985
..... .....    ...  ........                  ....        ....
..... .....    ...  ........                  ....        ....
----- -------- ---- ------------------ ---------- ---------
```

Sample data of the Company table are :

```
CO_ID CO_NAME
----- ------------------------------------------------
BUSINESS
------------------------------------------------------------
CITY                        STATE    CONT_TEL
--------------------------- -------- --------
00065 Philip Morris Asia Inc
```

```
Marketing - Retail Goods
Keene                    NH       25208657

00066 Procter and Gamble Hong Kong Limited
Marketing - Retail Goods
Marlow                   NH       25576586


00067 Sharp-Roxy (HK) Ltd
Marketing - Retail Goods
Acworth                  NH       33353688

00068 Wing On Co Ltd
Marketing - Retail Goods
Boston                   MA       39907230

00069 Basf China Limited
Marketing - Capital Goods
Salisbury                NH       38731500

00070 Electrolux (Far East) Ltd
Marketing - Capital Goods
Wolfeboro                NH       35886325

00061 Inchcape Pacific Limited
Marketing - Retail Goods
Sandown                  NH       37592832

00052 Lombard General Insurance Limited
Insurance and Actuary
Keene                    NH       38184640

00053 National Mututal Insurance Company Ltd
Insurance and Actuary
Bennington               MA       35337313

00054 New Zealand Insurance Company
Insurance and Actuary
Winnisquam               NH       31734033

00055 Manulife Financial
```

```
Insurance and Actuary
Jefferson                NH       37812548

00056 Nippon International Underwriters Agency Ltd
Insurance and Actuary
Sanbornville             CT       26330036

00057 The Prudential Assurnace Company
Insurance and Actuary
Munsonville              NH       37809464

00058 Hitachi Asia (Hong Kong) Ltd
Marketing - Retail Goods
Farmington               MA       35837651

00059 Hong Kong Seibu Enterprise Co Ltd
Marketing - Retail Goods
Marlow                   NH       20127784

00060 The Dairy Farm Company Limited
Marketing - Retail Goods
Wolfeboro                NH       23336253

00051 Heath Hudig Langeveldt Ltd
Insurance and Actuary
Salisbury                NH       32100701
..... ......................................
.........................................
..................... ........  ............
-----------------------------------------------
```

Sample data of the Position table are :

```
CO_ID PO_NAME                                          SE  START_AGE
----- -------------------------------------------------- -- ----------
   END_AGE
----------
00132 Building Services Engineer                       U          22
        23

00150 Building Services Engineer                       U          21
        23
```

```
00156 Building Services Engineer                U    20
      24

00168 Building Services Engineer                U    20
      23

00176 Building Services Engineer                U    21
      25

00177 Building Services Engineer                U    21
      24

00040 Certified Public Accountant               U    28
      33

00041 Certified Public Accountant               U    29
      33

00122 Certified Public Accountant               U    30
      33

00150 Certified Public Accountant               U    29
      33

00195 Certified Public Accountant               U    30
      32

00198 Certified Public Accountant               U    29
      33

00036 Chartered Engineer                        U    22
      28

00038 Chartered Engineer                        U    23
      26

00040 Chartered Engineer                        U    23
      27

00071 Chartered Engineer                        U    23
```

```
      28

00118 Chartered Engineer                                    U          23
      27

00136 Chartered Engineer                                    U          23
      28

00176 Chartered Engineer                                    U          23
      27

00015 Chemical Engineer                                     U          17
      38

00194 Chemical Engineer                                     U          17
      37

..... ..................................................... .........     ...
.............
------------------------------------------------------------------------
```

Sample data of the Salary table are :

```
CO_ID PO_NAME                                       YEARSOF_EXP     SALARY
----- --------------------------------------------- ----------- ----------
00181 Actuary                                               5       24693

00181 Actuary                                               6       26721

00181 Actuary                                               7       28830

00196 Actuary                                               2       21109

00196 Actuary                                               3       22551

00196 Actuary                                               4       24493

00196 Actuary                                               5       26258

00003 Advertising Manager                                   5       22152
```

00003	Advertising Manager	6	23615
00003	Advertising Manager	7	25600
00003	Advertising Manager	8	26866
00003	Advertising Manager	9	28144
00003	Advertising Manager	10	29436
00037	Advertising Manager	5	18616
00037	Advertising Manager	6	20401
00037	Advertising Manager	7	22257
00037	Advertising Manager	8	23722
00037	Advertising Manager	9	25216
00038	Advertising Manager	5	22042
00038	Advertising Manager	6	23503

Sample data of the School table are :

SCHO	SCHOOL_NAME	CITY	STATE
AU	American University	Washington	DC
BATE	Bates College	Lewiston	ME
BOWD	Bowdoin College	Brunswick	ME
CALT	Cal. Institute of Tech.	Pasadena	CA
COLB	Colby College	Waterville	ME

DREW	Drew University	Madison	NJ
FLU	University of Florida	Gainesville	FL
HVDU	Harvard University	Cambridge	MA
MIT	Mass. Institute of Tech.	Cambridge	MA
PRDU	Purdue University	West Lafayette	IN
QUIN	Quinnipiac College	Hamden	CT
STAN	Stanford University	Stanford	CA
UME	University of Maine	Orono	ME
USCA	U. of Southern California	Los Angeles	CA
YALE	Yale University	New Haven	CT
....	.....	.....	...

Sample data of the Speciality table are :

CO_ID	PO_NAME	FIELD
SUB_FIELD		
00001	Actuary	Business Admin
00007	Actuary	Business Admin
00023	Actuary	Business Admin
00031	Actuary	Economics
00012	Accountant	Accounting
00031	Actuary	Business Admin

```
00098 Actuary                                Economics
00098 Actuary                                Business Admin
00126 Actuary                                Economics
00131 Actuary                                Economics
00131 Actuary                                Business Admin
00152 Actuary                                Economics
00152 Actuary                                Business Admin
00171 Actuary                                Business Admin
00181 Actuary                                Business Admin
00136 Reporter                               Jouralism
00166 Reporter                               Jouralism
00185 Reporter                               Jouralism
00011 Structural Engineer                    Civil Engrg.
00019 Structural Engineer                    Civil Engrg.
00024 Structural Engineer                    Civil Engrg.
00067 Structural Engineer                    Civil Engrg.
00073 Structural Engineer                    Civil Engrg.
00079 Structural Engineer                    Civil Engrg.
..... ..................................................... ......................
.........................

------------------------------------------------------------------------------
```

Sample data of the required language table are :

CO_ID	PO_NAME	LAN_NAME
00194	Personnel Manager	English
00194	Quality Control Engineer	Japanese
00194	Quality Control Engineer	Mandarin
00194	Quality Control Engineer	Cantonese
00194	Quality Control Engineer	English
00194	Structural Engineer	Cantonese
00194	Structural Engineer	English
00195	Quality Control Engineer	French
00195	Quality Control Engineer	Mandarin
00195	Quality Control Engineer	Cantonese
00195	Quality Control Engineer	English
00196	Accountant	Cantonese
00196	Accountant	English
00196	Electrical Engineer	Japanese
00196	Electrical Engineer	Mandarin
00196	Electrical Engineer	Cantonese
00196	Electrical Engineer	English
00197	Advocate	Cantonese
00197	Advocate	English
00197	General Practice Surveyor	Cantonese
00197	General Practice Surveyor	English
00197	Land Researcher	Cantonese
00197	Land Researcher	English
00198	Actuary	Japanese
00198	Actuary	Cantonese
00198	Actuary	English
00198	Adminstrative Officer	Cantonese
00198	Adminstrative Officer	English
00198	Advertising Manager	Cantonese
00198	Architect	Cantonese
00198	Architect	English
00198	Management Consultant	Mandarin
00198	Management Consultant	Cantonese
.....	.....	.....

Sample data of the language table are :

```
APPLI LAN_NAME
----- ----------
00267 Cantonese
00267 English
00276 Cantonese
00276 English
00287 Cantonese
00287 English
00319 French
00319 Cantonese
00319 English
00345 Cantonese
00345 English
00354 Cantonese
00354 English
00358 Cantonese
00358 English
00359 Cantonese
00359 English
00369 Cantonese
00369 English
00374 French
00374 Mandarin
00374 Cantonese
00374 English
00405 Mandarin
00405 Cantonese
00405 English
00415 Cantonese
00415 English
00416 Cantonese
00416 English
00418 German
00418 Mandarin
00418 Cantonese
..... ..........................................................................
--------------------------------------------------------------------------------
```

Sample data of the `job history` table are :

```
APPLI CO_ID PO_NAME                                                START_DAT
----- ----- ---------------------------------------------------- --------
```

```
END_DATE  START_SALARY END_SALARY
--------- ------------ ----------
00245 00083 Media Producer                                    26-APR-90
03-APR-92         15738       21125

00245 00050 Insurance Claims Manager                          26-JUL-92
14-JUL-95         18008       28358

00246 00029 Quality Control Engineer                          04-MAY-88
24-JUL-93         12103       25616

00246 00037 Quantity Surveyor                                 04-JUL-93
02-JUL-95         15282       21759

00247 00173 Designer                                          04-AUG-80
08-JAN-84          5955       10720

00247 00159 Designer                                          04-FEB-84
23-OCT-86         10129       16373

00247 00020 Layour Artist                                     04-DEC-86
19-JUN-91         12615       23056

00247 00135 Art Director                                      04-JUL-91
25-JUL-95         17187       29727

00249 00119 Executive Officer                                 16-MAY-80
01-FEB-87          6844       14131

00267 00067 Adminstrative Officer                             08-MAY-78
05-JUL-85          4900       13637

00267 00196 Accountant                                        08-AUG-85
13-AUG-88          9241       14530

00267 00120 Personnel Manager                                 08-SEP-88
22-JUL-95         11625       24507

00276 00108 Adminstrative Officer                             06-MAY-88
27-JUN-94         13913       31962
```

```
00276 00172 Reporter                                                    06-AUG-94
06-JUL-95         24969       29086

00287 00172 Lawyer                                                      14-JUL-84
02-MAR-91          8379       20272
..... ..... ..... ..................................................................
.................. .... ............
------------------------------------------------------------------------------
```

Bibliography

[1] AWAD, E. M., AND GOTTERER, M. H. *Database Management.* Boyd & Fraser Publishing company, 1992.

[2] BELLMAN, R., AND ZADEH, L. "Decision-making in a fyzzy environment". *Management Science 17* (1970), 8141–8164.

[3] BEZDEK, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, 1981.

[4] BOSC, P., GALIBOURG, M., AND HAMON, G. "Fuzzy Querying With SQL: Extensions and Implementation Aspects". *Fuzzy Sets and Systems 28* (1988), 333–349.

[5] BOSC, P., AND PIVERT, O. "Some Algorithms for Evaluating Fuzzy Relational Queries". In *Uncertainty in Knowledge Bases* (1990), pp. 430–442.

[6] BOSC, P., AND PIVERT, O. "About Equivalences in SQLf, A Relational Language Supporting Imprecise Querying". In *Fuzzy Engineering towars Human Friendly Systems IFES'91* (1991).

[7] BOSC, P., AND PIVERT, O. "SQLf: A Relational Database Language for Fuzzy Querying". *IEEE Transactions on Fuzzy Systems 3*, 1 (February 1995), 1–17.

[8] BOSC, P., PIVERT, O., AND FARQYHAR, K. "Integrating Fuzzy Query into an Existing Database Management System: An Example". *International Journal of Intelligent Systems 9* (1994), 475–492.

[9] BUCKLES, B., AND F.PETRY. "A fuzzy model for relation database". *Fuzzy sets and systems 33* (1982), 213–226.

[10] BUCKLES, B., AND PETRY, F. "Fuzzy databases and their applications". In *Fuzzy Information and Decision Processes*, M. M. Gupta and E. Sanches, Eds. North-Holland, Amsterdam, 1982.

[11] BUCKLES, B., AND PETRY, F. "Query language for fuzzy databases". In *Management Decision Support Systems Using Fuzzy Sets and Possibility Theory*, J.Kacprzyk and R. Yager, Eds. Verlag TUV Reinland, Koln, 1985.

[12] CHANG, C. "Decision support in an imperfect world". *Research Report of IBM San Jose RJ3421* (1982).

[13] FODOR, J., AND ROUBENS, M. *Fuzzy Preference Modelling and Multicriteria Decision Support.* Kluwer Academic Publishers, 1994.

[14] G.GRAHAM, I., AND JONES, P. L. *Expert Systems-Knowledge, Uncertainty and Decision.* Chapman and Hall, 1988.

[15] GUPTA, M. M., SARIDIS, G. N., AND GAINES, B. R. *Fuzzy Automata and Decision Processes.* North-Holland, Amsterdam, 1977.

[16] GUPTA, M. M., AND YAMAKAWA, T. *Fuzzy Logic in Knowledge-Based Systems,Decision and Control.* North-holland, 1988.

[17] ICHIKAWA, T., AND HIRAKAWA, M. "ARES: a relational database with the capability of performing flexible interpretion of queries". *IEEE Transaction on Software Engineering 12* (1986), 624–634.

[18] ISHIZUKA, M., FU, K. S., AND YAO, J. T. P. "SPERIL: An expert system for damage assessment of existing structure". *Proc. of the Sixth Internationnal Comference on Pattern Recognition* (1982).

[19] KACPRZYK, J., AND ORLOVSKI, S. A. *Optimization Models Using Fuzzy Sets and Possibility Theory.* D.Reidel Publishing Company,Dordrecht, 1987.

[20] KACPRZYK, J., ZADROZNY, S., AND ZIOLKOWSKI, A. "FQUERY III+: A human consistent database querying system based on fuzzy logic with linguistic quantifiers". *Information systems 14* (1989), 443–453.

[21] KACPRZYK, J., AND ZIOLKOWSKI, A. "Database queries whth fuzzy linguistic quantifiers". *IEEE Transactions on Systems, Man, and Cybernetics 16*, 3 (1986).

[22] KAMEL, M., AND HADFIELD, B. "Fuzzy Query Processing Using Clustering Techniques". *Information Processing & Management 26*, 2 (1990), 279–293.

[23] KAUFMANN, A., ZADEH, L. A., AND SWANSON, D. L. *Theory of Fuzzy Subsets.* Academic Press, New York, 1975.

[24] LAI, Y.-J., AND HWANG, C.-L. *Fuzzy Mathematical Programming Methods and Applications.* Springer-Verlag, 1992.

[25] LAI, Y.-J., AND HWANG, C.-L. *Fuzzy Multiple Objective Decision Making Methods and Applications.* Springer-Verlag, 1994.

[26] LEUNG, K. S., AND LAM, W. "Fuzzy concepts in expert systems". *IEEE Computer 21* (1988), 43–56.

[27] LEUNG, K. S., WONG, M. H., AND LAM, W. "A Fuzzy expert database systems". *Data and Knowledge Engineering 4* (1989), 287–304.

[28] LI, D., AND LIU, D. *A Fuzzy Prolog Database System.* Research studies press ltd., 1990.

[29] MANSFIELD, W. H., AND FLEISCHMAN, R. M. "A high-performance, ad hoc, fuzzy query processing system". *Journal of Intelligent Information systems 2* (1993), 397–419.

[30] MIYAOMOTO. "Fuzzy control and its applications". *Systems and Control 25* (1986), 458–465.

[31] MOTRO, A. "VAGUE: a user interface to relational database that permits vague queries". *ACM Transactions on Office Information Systems 6* (1988), 187–214.

[32] OGAWA, H., FU, K. S., AND YAO, J. T. P. "SPERIL-II: An expert system for demage assessment of exidring structure". In *Appoximate Reasoning in Expert Systems*, M.M.Gupta, Ed. Elsevier North-Holland Science Publishers, 1985.

[33] RICHMOND, S. B. *Operations Research for Management Decisions.* The Ronald Press, 1968.

[34] SELIM, S. Z., AND ISMAIL, M. A. "Soft clustering of multidimensional data: a semi-fuzzy approach". *Pattern Recognition 17* (1984), 559–568.

[35] SHORTLIFFE, E. H. *"Computer-Based Medical Consultation: MYCIN".* American Elsevier, New York, 1976.

[36] SUGENO, M. *"Industrial Applications of Fuzzy Control".* North-Holland, 1985.

[37] TAHANI, V. "A conceptual framework for fuzzy query processing – a step toward intelligent database systems". *Information Processing and Management 13* (1977), 289–303.

[38] TAKAHASHI, Y. "A Fuzzy Query Language for Relational Databases". *IEEE Transactions on Systems Man. and Cybernetics 21*, 6 (Nov. 1991).

[39] TERANO, T., ASAI, K., AND SUGENO, M. *"Fuzzy Systems Theory and its Applications".* Academic Press, Boston, 1992.

[40] UMANO, M. "FREEDOM-0: a fuzzy database system". In *Fuzzy Information and Decision Processes*, M. M. Gupta and E. Sanches, Eds. North-Holland, Amsterdam, 1982.

[41] UMANO, M. "Retrieval from fuzzy database by fuzzy relational algebra". In *Proceedings of IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis* (Marseille, France, 1983), pp. 1–6.

[42] WONG, M. H., AND LEUNG, K. S. "A fuzzy database-query language". *Information systems 15* (1990), 583–590.

[43] ZADEH, L. A. "Fuzzy sets". *Information and Control 8* (1965), 338–353.

[44] ZADEH, L. A., AND KACPRZYK, J. *Fuzzy Logic for the Management of Uncertainty.* John Wiley & Sons, Inc., 1992.

[45] ZEMANKOVA-LEECH, M., AND KANDEL, A. *"Fuzzy Relational Data Bases – A Key to Expert Systems".* Verlag TUV Reinland, Koln, 1984.

[46] ZIMMERMANN, H. J. *"Fuzzy Set Theory and Its Applications".* Kluwer-Nijhoff, 1986.

[47] ZIMMERMANN, H. J. *Fuzzy Set Theory and its Applications.* Kluwer Academic Publishers, 1991.