in Proceedings of ACM SIGMOD '93, Washington D.C.

The Design and Implementation of CoBase *

W. W. Chu M. A. Merzbacher L. Berkovich

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

CoBase, a cooperative database, is a new type of distributed database that integrates knowledge base technology with database systems to provide cooperative (approximate and conceptual) query answering. Based on the database schema and application characteristics, data are organized into conceptual (type abstraction) hierarchies. The higher levels of the hierarchy provide a more abstract data representation than the lower levels. Generalization (moving up in the hierarchy), specialization (moving down in the hierarchy) and association (moving between hierarchies) are the three key operations in deriving cooperative query answers.

Relaxation in CoBase can also be specified explicitly in the query by the user or calling program through cooperative operators. We have extended SQL to CSQL by adding cooperative primitives. We describe the CoBase software implementation, including an inter–module data protocol that provides a uniform module interface. This modular approach provides flexibility in adding new relaxation modules and simplifies software maintenance.

CoBase uses LOOM as its knowledge representation and inference system and supports relational data bases (e.g. Oracle and Sybase). We have demonstrated the feasibility and functionality of CoBase on top of a Transportation Database. The CoBase methodology has also been adopted in the multimedia medical distributed database project at UCLA, which provides approximate query answers to medical queries.

*Supported by DARPA contract N00174–91–C–0107

1 Introduction

In a conventional database, if required data is missing, if an exact answer is unavailable, or if a query is not well-formed with respect to the schema, the database returns a null answer or an error. An intelligent system is much more resourceful and cooperative, permitting conceptual level queries (containing concepts that may not be expressed in the database schema) when the user does not know the precise schema, providing approximate answers when some data is missing, or even volunteering associative (relevant) information to the query answer. *CoBase* is an implementation of a new generation of distributed database system called cooperative databases which incorporate these intelligent capabilities.

In this paper we first present the *type abstraction hierarchy*, a data structure that provides a structured approach to query modification and relaxation by explicit cooperative operators. Next, we present the cooperative SQL primitives and selected examples. We then present the implementation of CoBase. Finally, we discuss the technology transfer of CoBase to transportation and medical database applications.

2 A Structured Approach

Although knowledge about the application domain can be expressed as a set of logical rules, such a rule–based approach lacks a systematic organization to guide the query transformation process [10, 13]. To remedy this problem, we use the notion of a *type abstraction hierarchy* (TAH) [4, 6, 7, 12] for providing an efficient and organized framework for cooperative query processing. Type abstraction emphasizes the abstract representation of object instances (see Figure 1). For example, "runway_length_range" in the domain of *range* (e.g. 4,000 to 8,000 ft.) is a more abstract representation than "runway length" in the domain of *length* (e.g. 6,000 ft). Abstractions of several attribute values can be *combined* to form abstraction of tuples. For example,

airport (location_area, runway_length_range,
runway_width_range, parking_area, ...)

is an abstract representation of

airport (location, runway_length,
runway_width, parking_space, ...)

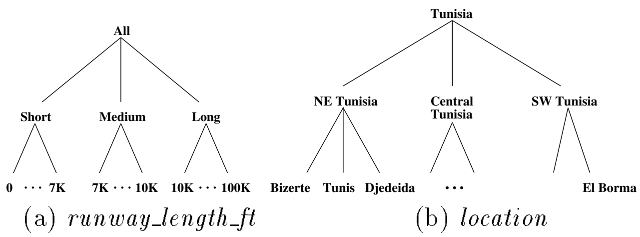

Figure 1: Type Abstraction Hierarchies

CoBase modifies a query by altering the query conditions. CoBase first selects one attribute value from the query condition for relaxation; this value is called the *target value*. The target value is located at the bottom of the type abstraction hierarchy for its attribute. CoBase uses *generalization* (moving up the TAH) and *specialization* (moving down the TAH) to modify the query and broaden the search space. Type Abstraction Hierarchies provide a structured way for query modification which is different from the rule-based approach [10]. Thus, our proposed structured approach to cooperative query answering is scalable to problems with large knowledge and data bases. We are currently researching how to generate TAH automatically from databases when given an application context and a set of queries [16, 8].

Although CoBase uses relaxation when no answer is found, query modification may also be specified explicitly by the user through a set of cooperative operations with a pre-specified relaxation range and specification. We have extended SQL to include these cooperative operations as well as the relaxation control operations to form a new cooperative query language, *CSQL*.

3 Cooperative SQL (CSQL)

3.1 CSQL Cooperative operators

The cooperative operations consist of the following four types:

- Context free operations
  - **approximate operator** ($\wedge$)
    relaxes the specified values within the approximate range that is predefined by the user. For example, $\wedge$9AM will be changed to the interval (8am, 10am). Further, (3pm, $\wedge$5pm) will be replaced by (3pm, 6pm).
  - **within**
    specifies a set membership for an attribute value. For example, *airport* **within** {'LAX', 'Burbank'}
- Context sensitive operations
  - **near-to**
    specification of geographical nearness. For example, **near-to** 'Bizerte' requests a list of cities located within a certain Euclidean distance from 'Bizerte'. The distance is context–dependent, so it might be 50 miles for airports, but only 5 miles for restaurants.
  - **similar-to**
    requests a set of objects semantically similar to the given object. Similarity is a multiple attribute operator with a weight assigned to each attribute in accordance with the relative importance of that attribute. Each object fulfilling the query conditions is evaluated against the given object by calculating the weighted mean-squared error over the list of attributes. The similar objects are listed in ranked order, along with their measures.
- Control Operators:
  - **relaxation–order**
    specifies the attribute relaxation order. For example, **relaxation-order** *(runway_length runway_width)* indicates that if no exact answer is found, then *runway_length* should be relaxed first. *runway_width* is relaxed only if no answer is found after relaxing *runway_length*.
  - **not-relaxable**
    specifies the attributes that should never be relaxed. For example, **not-relaxable** *(location_name)* indicates that condition clauses containing *location_name* may not be relaxed by CoBase.
- User/System interaction operators
  - **nearer, further** provide the user with ability to control the relaxation range of **near-to** interactively. **Nearer** reduces the distance by a pre-specified percentage, while **further** increases the distance by a pre-specified percentage. The percentage change can be specified by the user with a default of 50%.

3.2 Examples

We now present a few selected examples that illustrate the capability of the cooperative operators. The examples use two relations from the transportation planning domain, *geoloc* and *airports*. *geoloc* contains geographic information, such as latitudes and longitudes, while *airports* contains airport specifications. The two relations can be joined on the shared key attribute *geo_code*. We use the TAHs in Figure 1 for query modification, and the distance contexts from in Table 1.

Attribute	Default Distance
Restaurant	5 miles
City	200 miles
Airports	50 miles

Table 1: Distance Contexts for **near-to** Operator

CSQL Query 1:
List the airports with the parking capacity *approximately* equal to 200,000 square feet.

select *airport_name, parking_sq_ft*
from *airports*
where *parking_sq_ft* = $\wedge$200,000

This CSQL query is translated by CoBase into the following SQL:

select *airport_name, parking_sq_ft*
from *airports*
where *parking_sq_ft* >= 100,000 and
parking_sq_ft <= 300,000

CSQL Query 2:
Find all the cities located *near to* Bizerte.

select *location_name*
from *geoloc*
where *location_name* **near-to** 'Bizerte'

The **near-to** operator is context sensitive. According to the context, the system preassigns an appropriate range in Euclidean distance for the operator. In this case, the range for cities is 200 miles (Table 1). Therefore, all the cities located within 200 miles from Bizerte are retrieved and inserted into the query. CoBase then executes the new query:

select *location_name*
from *geoloc*
where *location_name* **within** { 'Bizerte', 'Djedeida', 'Gafsa', 'Gabes', 'Sfax', 'Sousse', 'Tabarqa', 'Tunis' }

The user can adjust the range for the **near-to** operator by using the **nearer** or **further** primitive. For example, if the user applies the **nearer** operation to reduce the distance to 100 miles, CoBase retrieves the new set of cities, modifies the query, and executes:

select *location_name*
from *geoloc*
where *location_name* **within** { 'Bizerte', 'Djedeida', 'Tabarqa','Tunis' }

CSQL Query 3:
List all the airports with the runway length greater than 7500 feet and runway width greater than 100 feet. If there is no answer, *relax the runway length condition first.*

select *airport_name, runway_length_ft,*
    runway_width_ft
from *airports*
where *runway_length_ft* > 7500
and *runway_width_ft* > 100
relaxation-order *(runway_length_ft,*
    runway_width_ft)

Based on the relaxation order, CoBase uses the TAH for runway length to relax the query:

select *airport_name, runway_length_ft,*
    runway_width_ft
from *airports*
where *runway_length_ft* > 7000
and *runway_width_ft* > 100

If this new query yields no answer, then CoBase continues by relaxing *runway width.*

CSQL Query 4:
Find the airports in Tunisia *similar to* the Bizerte airport. Use the attributes *runway_length_ft* (with weight 2.0) and *runway_width_ft* (with weight 1.0) as the criteria for similarity. Provide the best n answers.

select *airport_name*
from *airports, geoloc*
where *airport_name* **similar-to** 'Bizerte' **based-on**
((runway_length_ft 2.0) (runway_width_ft 1.0))
best n
and *country_state_name_long* = 'Tunisia'
and *geoloc.geo_code* = *airports.geo_code*

CoBase first finds all the airports in Tunisia, dropping the **similar-to** clause.

select *airport_name*
from *airports, geoloc*
where *country_state_name_long* = 'Tunisia'
and *geoloc.geo_code* = *airports.geo_code*

Based on the attributes *runway_length_ft* and *runway_width_ft* and their corresponding weights, CoBase compares each of the query answers with 'Bizerte' by computing the weighted mean squared error [17]. The top n (a pre-specified answer set size) answers, ranked by similarity, are presented.

4 Implementation

The Cobase control flow chart is shown in Figure 2. When a query is presented to CoBase, the system first relaxes any explicit cooperative operators in the query. The modified query is then presented to the underlying database system for execution. If no answers are returned, then the cooperative query system, under the direction of the *Relaxation Manager*, based on the TAHs, user profile and application context, relaxes the queries by query modification. The relaxed query is executed, and, if there is no answer, the relaxation manager will broaden the search scope until an answer is found. The relaxation manager can change its relaxation behavior based on explicit user requirements such as **not-relaxable** and **relaxation-order**.

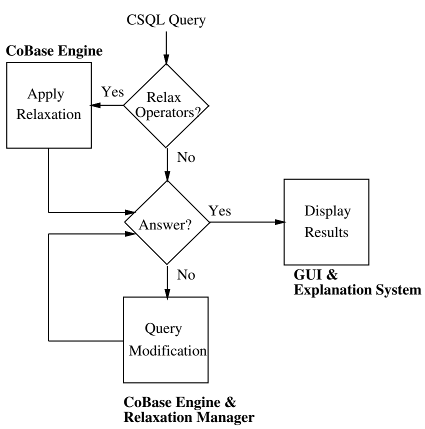

Figure 2: CoBase Flow Chart

Figure 3 shows an overview of the CoBase architecture. CoBase stores the Type Abstraction Hierarchy and relaxation ranges for the explicit cooperators in a LOOM [14] knowledge base. When CoBase asks queries, it can either use the Loom Interface Module (LIM) [15], or directly query the underlying relational database (e.g. Oracle and SyBase).

CoBase's control mechanisms include a graphical user interface (GUI), Explanation System and Relaxation Manager. The Relaxation Manager controls the search for approximate answers by directing the individual relaxation modules. Each relaxation operation is implemented as a separate software module.

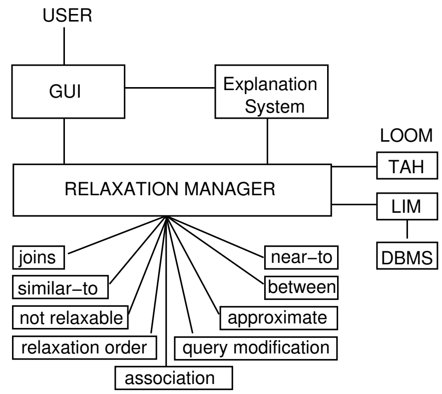

Figure 3: CoBase Architecture

Such a highly modular approach allows easy expansion, maintenance and modification to the system.

The Explanation System provides explanations of how approximate answers were derived, along with information on semantic nearness when appropriate. A graphical user interface (GUI) displays the query, query results, TAHs and explanations of the relaxation processes.

4.1 CoBase Inter–module Protocol

A tuple-based data structure allows communication between the Relaxation Manager and the individual relaxation modules. The tuple consists of four elements for each condition:

relation name
attribute name
value
comparison operator

For example, the query condition, $runway\_width\_ft >$ 7000 is represented by the tuple

(*airports* , *runway_width_ft* , *7000* , >)

The relaxation modules modify the tuple based on the context information (Table 1) or the TAH (Figure 1) and the *value* can be a list in the case of the set-membership operators. Using the CSQL example 2 from section 3.2, the **near–to** module transforms the tuple (*geoloc*, *location_name*, Bizerte, **near–to**) into (*geoloc*, *location_name*, {Bizerte, Djedeida, ...}, **within**) and returns the result to the Relaxation Manager.

The tuple data structure provides a uniform data interface for cooperative modules. The four elements in the tuple, along with the TAH and context information are sufficient to allow the modules to communicate with one another and with the Relaxation Manager, thus providing flexibility in adding new relaxation primitives to the system. Further, the interface allows relaxation module modification to be carried out independently, thus greatly simplifying the task of software maintenance.

4.2 Association

Association in CoBase is a multi-step post-process. After the query is executed, the answer information is gathered with the query conditions, user profile and application constraints. This joint information is matched against rules from the knowledge base to identify relevant associative information [5]. The rules can include any CoBase construct, such as conceptual conditions (e.g. *runway_length_ft* = **short**) or explicit cooperative operations (*city* **near-to** 'Bizerte'). For example, consider the query

select *name, runway_length_ft*
from *airports*
where *runway_length_ft* $>$ 6000

Based on the joint information, associated information on runway conditions and weather for the corresponding airports is retrieved from the knowledge base. The associated information is appended to the query answer, as shown in Table 2.

Query Answer		Associated Info	
name	*runway–_length_ft*	*runway condition*	*weather*
Jerba	9500	Damaged	Sunny
Monastir	6500	Good	Foggy
Tunis	8500	Good	Good

Table 2: Query Answer and Associated Information for the Selected Airports

5 Technology Transfer

CoBase uses LOOM as knowledge representation and inference system and supports relational data bases (e.g. Oracle and Sybase) and LIM (the Loom Interface Module) [15, 11]. Access to distributed databases is made possible through SIMS, a transparent multi-database access layer [1]. We have demonstrated the feasibility and functionality of CoBase on top of the Transportation Database and it is a part of the Data Management provision of the Transportation Planning Initiative.

In medical databases that store X–ray and MR images, images need to be retrieved by object feature or contents rather than patient ID. The queries asked are often conceptual and not precisely defined. We need to use knowledge about the application (e.g. age class, ethnic class, disease class, bone age etc.), user profile and query context to answer such queries [9]. Further, exact matching of features is very difficult, if not impossible. For example, if the query "Find the treatment methods used for tumor x on 12-year-old Korean males," cannot be answered, based on the TAHs for tumors, age, and ethnic groups, we can relax "tumor x" to "tumor class X", and "12-year-old Korean male" to "pre-teen Asian," which results in the relaxed query, "Find the treatment methods used for the tumor class X on pre-teen Asians." Further, we can obtain such relevant information as the *success rate, side effects,* and *cost of the treatment* from the association operations. We have applied the CoBase

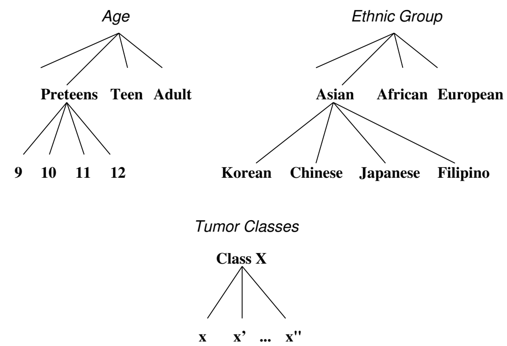

Figure 4: TAHs for the Medical Example

technology to the medical imaging data base [3, 2]. The TAHs (see Figure 4) are constructed based on the available application domain and medical knowledge available in the textbooks and/or literature. Once the TAH for the medical domain and application have been constructed, query relaxation and modification can then be carried out in the same manner as in the transportation domain.

6 Conclusion

We have presented a structured approach that uses *type abstraction hierarchy* for providing query modification. Such an approach provides multi-level knowledge representation, and is also scalable to larger systems. We have also extended SQL to CSQL, adding cooperative operators that allow the user to explicitly specify relaxation ranges and relaxation control. Based on user profile and application contexts, the relaxation manager controls the search by a well–defined data interface to individual cooperative modules. One of these modules is responsible for finding associative (relevant) information. Our design is modular, allowing for easy addition of new modules as well as simple maintenance of existing modules. An explanation facility is included which summarizes the query modifications and relaxation process, and also provides the nearness of the approximate answer to the exact answer. CoBase is part of the transportation planning initiative, has also been implemented on a medical imaging database and has received favorable responses from the medical community.

7 Acknowledgements

The research and development of CoBase has been a team effort. We would like to acknowledge Qiming Chen, Kuorong Chiang, Michael Minock and Berthier Ribeiro for their contributions in design and development, and Gladys Chow and Jason Robbins for their implementation efforts.

References

[1] Y. Arens and C. Knoblock. Planning and reformulating queries for semantically–modelled multidatabase systems. In *Proceedings First International Conference on Information and Knowledge Management (CIKM)*, pages 92–101, Baltimore, Maryland, 1992.

[2] A. F. Cardenas, I. T. Ieong, R. Barker, R. K. Taira, and C. M. Breant. The knowledge–based object–oriented picquery+ language. *to appear in IEEE Transactions on Knowledge and Data Engineering*, August 1993.

[3] W. W. Chu, A. F. Cardenas, and R. K. Taira. A knowledge–based multimedia medical distributed database system — KMeD. Technical Report 93–005, UCLA Computer Science Department, 1993.

[4] W. W. Chu and Q. Chen. Neighborhood and associative query answering. *Journal of Intelligent Information Systems*, 1(3/4):355–382, 1992.

[5] W. W. Chu and Q. Chen. Pattern–based data and knowledge integration for intelligent query answering. *Heuristics, Internation Journal of Knowledge Engineering, special issue on Intelligent Information Systems*, 1993.

[6] W. W. Chu and Q. Chen. A structured approach for cooperative query answering. to appear in IEEE Transactions on Knowledge and Data Engineering, 1994.

[7] W. W. Chu, Q. Chen, and R. Lee. Cooperative query answering via type abstraction hierarchy. In S.M. Deen, editor, *Cooperating Knowledge Based Systems*, pages 271–292. North-Holland, Elsevier Science Publishing Co., Inc., 1991.

[8] W. W. Chu and K. Chiang. A distribution sensitive clustering method for numerical values. Technical Report 93–0006, UCLA Computer Science Department, 1993.

[9] W. W. Chu, I. T. Ieong, R. K. Taira, and C. M. Breant. A temporal evolutionary object–oriented data model and its query language for medical image management. In *Proceedings of the 18th VLDB Conference*, Vancouver, British Columbia, 1992.

[10] F. Cuppens and R. Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In *Proc. of the 2nd international conference on expert database systems*, 1989.

[11] T. W. Finin, D. P. McKay, and A O'Hare. The intelligent database interface. *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI)*, 1990.

[12] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform of cooperative answering. In *Proceedings First International Workshop on Nonstandard Queries and Answers*, Toulouse, France, 1991.

[13] A. Hemerly, M. Casanova, and A. Furtado. Cooperative behavior through request modification. Technical report, IBM Brasil, Brazil, May 1991.

[14] R. MacGregor. The evolving technology of classification-based knowledge representation systems. in Principles of Semantic Networks: Explorations in the Representation of Knowledge, 1991. J. Sowa, ed.

[15] D. P. McKay, J. Pastor, and T. W. Finin. View–concepts: Knowledge–based access to databases. In *Proceedings First International Conference on Information and Knowledge Management (CIKM)*, pages 84–91, Baltimore, Maryland, 1992.

[16] M. Merzbacher and W. W. Chu. Instance-based clustering for databases. In *ASIS Third Workshop on Classification Research*, pages 101–114. American Society for Information Science, October 1992.

[17] A. Motro. FLEX: A tolerant and cooperative user interface to databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):231–245, 1990.