Transforming Databases for Experts

Jussi Stader, Robert Inder [1] and Paul W. H. Chung [2]

August 1993 128

in Proceedings of the Seventh International Conference on Industrial and Engineering Applications of AI and Expert Systems; Gordon and Breach, June 1994; pp 283–289.

[1] previously at AIAI, now at Human Communication Research Centre (HCRC) The University of Edinburgh

[2] previously at AIAI, now at Department of Chemical Engineering, Loughborough University of Technology

Abstract

Recognising and gathering relevant data is often a very time-consuming part of an expert's work. Databases can make the necessary information available, but current access methods require database skills which few experts have. Bridging this gap between experts and the information they require can greatly improve the experts' effectiveness.

This paper describes a system which uses knowledge-based techniques to allow users to browse databases without requiring database skills.

An object-oriented model of the domain, specified by domain experts, is used to determine how the content and structure of relational databases should be presented to the user. The system uses this model to ensure that information is presented in a form that is readily appreciated, and that familiar terms and concepts can be used to formulate queries, to control display of information, and to navigate through the database. Fuzzy matching techniques further support the user in specifying what information is required and they provide a more intuitive access to data.

This paper describes the implementation of a system embodying these ideas. The system has already been applied to a number of different databases in different domains.

1 Introduction

The value of data, and the importance of ensuring that it is conveniently available, motivates much of the use of computers. Storing and retrieving data is an archetypal application, and falling storage costs and rising display capabilities together allow an increasingly large range of data to be held on-line.

The majority of those who use data do so for routine purposes: warehousemen load lorries, students attend exams, customers check and pay their bills. These people have fixed data requirements, and thus their needs are straightforward. However, many other people carry out complex or creative tasks on the basis of substantial amounts of data. Initially, the work reported here focussed on geologists concerned with using exploration and production data to assess the likely value of looking for oil in one area or another, but there are others who have generally similar data access requirements: e.g. senior managers, system designers, researchers of almost any kind.

Much of the expertise of these people lies in their knowledge of the data they are using—its location, how to analyse it, the limitations on its accuracy or applicability—coupled with their ability to interpret it, and to see its significance to the activities and objectives of the organisation.

As they work, they often want to see additional data to continue or support the train of thought triggered by the data they have already retrieved. This makes it impossible to anticipate their data requirements, and thus to prepare interfaces to present the data they will need: the experts need to have the ability to access the information they require directly and control its display themselves.

Unfortunately, expertise in data does not make a database expert. Knowledge of what data is held is not enough to understand how it is held. For efficiency and compatibility reasons database structures do not usually reflect the structures of the real world, or the way experts think about the information.

Current developments in database technology, such as object-oriented or deductive databases, offer some gains in user friendliness and flexibility. But even if these technologies become standard, there will still be serious problems associated with converting existing data. There are tools for building interfaces to shield users from database details, they rely on being able to anticipate the queries and paths through the data that users will require, and are not flexible enough to allow for creative data browsing.

The approach described in this paper uses knowledge based techniques to provide a flexible tool for accessing information held in conventional databases without requiring database skills.

The system incorporates a model of the domain of the subject matter of the database which has been derived from experts in the area. This model is used to decide how the database structure and contents can be presented to the user so that it will be readily understood. Knowledge about the domain of expertise is also used for adapting the interface to the nature of the information, making interaction with the system more effective.

Fuzzy matching is used to soften the "all or nothing" approach to matching. It addresses issues of uncertainty, which can arise either because the user does not know how best to describe the required set of objects, or because the data itself is inaccurate. Vague or unreliable data in the database can make other data inaccessible and thus cause related information to be missed. Fuzzy matching techniques make it possible to deal with the such data and thus exploit all available information by returning not only items that match the specification exactly, but also near misses, i.e. items that are similar to the specification. This provides the means to relax constraints and thus gives users more flexibility in specifying queries and more control over the number of results a query produces.

2 Interface

The user interface for the system is based on the metaphor of *bags and viewers*, which is described in [Inder and Stader].

The user initiates an interaction with the database by specifying a type of object of interest and then providing constraints on the values of its attributes. These can be "fuzzy", thus allowing users to avoid worrying about the precise scope of their interests.

Constraints are created and modified by using *constraint editors*—pieces of software tailored to handling constraints on the values of a specific type of attribute. Each constraint editor interacts through its own window, presenting an interface specialised for editing constraints of a particular kind. For a numeric attribute, this may take the form of a slider for setting a value (or indeed a pair of sliders, for setting a range of values), whereas an editor for an attribute that takes symbolic values might offer the various possible values as a menu or a hierarchy, and the editor for a positional attribute might present a map.

Once a set of constraints has been specified, the objects that satisfy them can be "fetched " from the database and presented on the screen as a *bag*. However, bags reveal nothing about the objects they "contain": to see something of them, the user must invoke one or more *viewers*.[3] Each viewer displays some information about the object (or objects) in a particular bag. Some viewers will simply present a "form"—a set of attribute labels and the values corresponding values positioned on the screen. Others may display maps, images, graphs, or any other diagram which is specific to the domain of the data. These differences are all encapsulated within the viewer: for the rest of the system, each viewer simply produces a requirement to retrieve from the database a particular set of attributes for the objects in the bag.

---

[3] In practice, the system normally invokes a default viewer with each new bag.

Viewers generally allow users to easily generate queries related to the data they are displaying—for instance by clicking on an object for which more data is required. This is sufficient to specify a complete query because the the system can supply most of the constraints from context (i.e. the viewer). As a result, once an initial set of data has been displayed, further querying is very simple: a single mouse click to indicate "tell me more about...."

Within the system, queries have two components: a set of attributes to be displayed, derived from the set of viewers associated with the bag, and a set of constraints on the attribute values of interest, which can be fully specified by the user or partially derived from the viewer supporting the query.

3 Knowledge Bases

To help users use a database about a given domain, the system needs to know about the domain, the database and the connection between the two. These are presented in three distinct data structures, since there may be different domains and multiple databases: each is described independently.

3.1 Domain Description

At the centre of the system there is a model of the user's field of expertise, the domain model, which allows the system to communicate with users in their own terms. It is specified by an expert of the domain and it is used by the system to formulate queries, to navigate through the database, and to visualise data.

The domain model is structured along standard object-oriented lines. It specifies the types of objects that exist, their attributes (parameters), and the kinds of relationships between them. Attributes are also organised into a type hierarchy, so that information about attribute types can be used—for example, in specifying normal values, possible values and possible units.

The model should be structured to reflect the way experts themselves perceive their domain, and should not attempt to reflect the kinds of transformations that are often applied to databases to improve their efficiency or maintainability. For, while experts cannot be expected to be database specialists, they *can* be expected to know their area of expertise, and to relate to abstract models of that area. Thus the domain model is a suitable vehicle for interaction. Objects, attributes and relationships replace database concepts like tables, columns and joins between tables. The domain model is used to structure information and to determine the interface to the system, but it also provides the knowledge needed to support the user during interaction.

A graphical editor is used to generate domain descriptions. The models are specified by drawing diagrams from which specifications for the object system are generated automatically. This approach supports both the specification and the visualisation of objects, their attributes and the relationships between objects.

3.2 Database Description

For the system to query a database, it has to have a description of it which indicates both its structure and the semantics of the data it contains.

The database itself is described in terms of tables, columns and data types. Most relational database management systems, like Oracle, contain the relevant information in a *data dictionary* which can be used to automatically extract the database description needed.

The mappings between the database description in terms of tables and their columns, and the domain description in terms of objects and their attributes are specified with the help of *provisions*. They allow the object-oriented domain model to be filled with information from the database by specifying which parts of the database to access and how to derive the values of domain attributes. Provisions are always specific to an object's attribute, and they always refer to a specific database.

There are five basic types of provisions :-

1. Simple provisions specify a column in a table to access. The data in that column is used unchanged for the attribute's value. It appears in the domain as it is in the database.

2. Constant provisions are used where concepts in the domain model are fixed in the database. For example, if the database only contains information about wells drilled in the UK, then the country of wells will always be "UK". No database access is needed to determine this information.

3. Translation provisions allow complex transformations between database values and attribute values. Translation provisions can be used, for example, to combine the values of several different database columns (possibly in different tables) in order to produce or infer an attribute value. Another example for using translations is the interpretation of data, where implicit structure in textual data is used to extract information.

   There are no restrictions as to how complex translation provisions can be.

4. Lookup tables can be used for simple translations between database values and attribute values. This is useful for things like expanding abbreviations used in the database, e.g. "OG" becomes "oil and gas", or converting qualitative numbers into symbolic values, e.g. values 1 to 5 are translated into tiny, small, medium, large, and huge.

5. Indirect provisions allow attributes to be specified in terms of other attributes. The provisions of those other attributes are invoked when indirect provisions are mapped. At this level, provisions act as a user-programmable rule language for specifying data manipulation.

Attribute provisions include unit specifications. Units of values can be specified to be constant in the database, they can be fetched from the database, or even derived.

It is possible to define multiple provisions for a single attribute. When values are mapped during query processing the system chooses the most appropriate provision available.

Relationships between objects are mapped into database terms with the help of *relators*. Relators are the provisions of relationships. They are expressed in terms of constraints on database columns. Most relationships are supported in the database by the database designer, implemented through foreign keys. In these cases the relators simply specify a relational join of tables. For example, the interest of a well can be found via the well number in the interest table. However, some relationships are not directly supported in the database design. For example, to find the sedimentary layer above the bottom of a well, the depth information of the layer has to be compared to the depth of the well.

A point-and-click style editor supports the specification of mappings between the domain and database descriptions. The editor simplifies the complex task of connecting these two models whose structures can differ substantially. The editor offers default provisions to speed up the bulk of the mapping specifications and it points out areas where mappings are not yet specified.

4 Fuzzy Matching

In addition to wanting data on things with precisely specified attributes, someone using a database to support creative tasks will often want information on things with certain general properties, or even on things that are "like" a given object—true query by example.

It is easy to provide facilities for users to specify limits on the acceptable values for different attributes, and thereby spread the focus of their queries. However, such an approach obliges users to consciously sharpen their criteria for acceptability, which is not always what they want to do. More importantly, such mechanisms do not handle "trade offs" – if the values of some attributes are very close to their "targets", greater variation can be tolerated in others. An ideal data retrieval system should support users in formulating imprecisely specified queries, accepting them and handling borderline cases "intelligently".

A fuzzy matcher is expected to match a specification to a collection of data, and return information not only about items that precisely satisfy the specification, but also about items which are sufficiently similar to be of interest. In this section we refer to the value of an attribute that the user specified should be searched for as the target, denoted t, and to the value actually stored for that attribute as the data, d. The fuzzy matcher must be able to determine the acceptability of presenting something with value d in response to a query specifying t. This will be characterised by a number in the interval [0,1], indicating the *goodness of fit* (GOF) of the target and the data.

To determine the GOF we use the idea of a *characteristic function* (CF) from fuzzy set theory (see [Gaines] for more details). The CF for an attribute, $gof(t, d)$, returns the goodness of fit between t and d. The key aspects of the behaviour of the CF required for handling different kinds of data types are now considered. For a more detailed discussion of fuzzy matching in databases see [Chung and Inder].

4.1 Matching with Numbers and Numeric Ranges

The behaviour of a CF for matching a target t against a desired data value d should be such that as d moves away from t, the value of the GOF decreases. The simplest such function would decrease linearly with the distance between t and d, falling from a maximum of 1 and clipped at a minimum of 0. More complex functions could be used, for instance to remove the sharp cut-off point or to shift emphasis to very near misses.

If the target value t represents an upper bound then the $gof(t.d)$ should obviously be 1 if d is less than or equal to t, and should decrease as d increases above t. If t represents a lower bound then the expected behaviour of the CF is the reverse of the above.

When handling ranges, we need to recognise that there are two types of range: extent ranges and interval ranges. These have quite different meanings, and they have to be interpreted differently when matching against them.

The first implies a value with an extent between the limits [min, max]. For example, a sandstone stretches from 1500 to 2000 ft deep. The second implies an interval between min and max where the actual value may lie. For example, the porosity of a rock is between 5 to 8%. There is only one value. The use of interval is necessary because of the lack of confidence in providing a precise value.

A fuzzy matcher needs to be able to distinguish between these two types of ranges and to handle them efficiently. For a specified target extent range we are interested in knowing how much of the target has been satisfied. For example, a query that requests basins that have sandstone between a depth of 10000 and 11000 ft can equally well be satisfied by a basin that has sandstone between a depth of 9000 to 12500 ft and by another basin that has sandstone extending exactly between 10000 and 11000 ft. In this case, as long as the target is satisfied it does not matter if part of the data range is outside the target range. On the other hand, if a basin has sandstone only between 10500 and 10900 ft then it has only partially satisfied the target.

As for a specified target interval range, as opposed to extent range, we are interested in how much of the data actually falls inside range. For example, a query that requests reservoir porosity between 8 and 10% is better satisfied by the data range 8 to 10% then the range 7 to 11%, but 9 to 9.5% would be just as good.

In consider the GOF of two ranges, there are two relevant factors: the degree of overlap between them and their central offset. The former is more important, the latter is necessary only if we need a very sensitive GOF measure. For more details see [Chung and Inder].

4.2 Matching With Symbols

There are two types of symbols that a fuzzy matcher has to handle. The first we shall call qualitative symbols. There are adjectives describing qualitative values. Often an expert will specify a qualitative rather than a quantitative value—e.g. a *deep* well. A fuzzy matches should be able to match such qualitative symbols not only against one another, but also against actual numeric values. This is handled using the concept of linguistic variables found in fuzzy logic [Schmucker]. Associated with a linguistic variable is a set of linguistic values. For example, let TOC be a linguistic variable. The set of linguistic values related to it may be: very low, low, moderate, high, very high.

In this set the primary terms are: low, moderate and high. Each is associated with a CF that indicates how well a numerical TOC value can be associated with that term – i.e. the GOF between the number and the category. The word "very" is a hedge. This means apply a modifier function to the GOF value of the primary term.

The second type of symbol is referred to as object symbol. These are labels for classes of objects like "sandstone" and "shallow marine environment", and they cannot be readily ordered. A fuzzy matcher must determine how similar two such objects are according to some criteria. This could be done by explicitly specifying a table of GOF values, rather like an inter-city mileage chart, although this tends to require unacceptable amounts of effort to check and specify for large sets of symbols. Alternatively, the symbols can be grouped into a classification structure, with the GOF of two objects being determined by the "weighted distance" between them in the hierarchy.

5 Queries

Queries specified by the user in terms of objects, attributes and attribute values are translated into database terms using the provisions described in section 3.2. The query constraints are "pushed through" the provisions by performing the specified mappings, producing one or more constraints on the database items that are the source of the provision.

Note that for most provisions only one mapping specification is needed from which transformations in both directions can be performed: from domain values to database values for constraints and from database values to domain values for results. However, for the complex translation provisions both directions have to be specified separately (consider a constraint on an attribute value which is derived by the calculation `attribute = column1 + column2`).

Where provisions contain indirect mappings the translated constraints will still contain references to domain attributes and thus the process has to be repeated. When all constraints are expressed in terms of tables, columns and database values, the query text (SQL) can be generated.

During query generation the processing of results is prepared: parsing of results is supported by determining the types of expected result values, and translation of database results into domain terms is prepared by determining which mappings and conversions are to be performed on data fetched from the database and generating the code to translate results. When the results are passed back from the database they are processed by executing the generated code to apply the mappings and conversions to each match in turn.

Unit conversions are performed whenever required during query generation and result processing. Conversions can be numerical (feet to metres) or symbolic (fine-grained geological ages to coarse-grained ones), or a mixture (ages in years to named geological epochs).

Fuzzy constraints are handled in two stages. First, the query is pre-processed to turn each fuzzy constraint into a precise constraint which will include every object that matches on that attribute with GOF values above a threshold. Then, when objects have been retrieved, the GOF of each attribute is calculated using the appropriate characteristic function. These are combined to produce an overall value for the whole match, and only objects for which this exceeds the threshold are included in the bag.

6 Implementation Issues

The system described above has been implemented in Prolog and tested using a number of different (pre-existing) databases built in Oracle.

6.1 Architecture

The running system consists of three separate processes: the window manager, the database manager and the system engine which is in control. Commands to the two managers are issued via pipes using restricted protocols, and files are used for passing query texts and results between the system engine and the database manager. Thus the window manager and the database manager are independent of the system, which is particularly important for the database manager: it does not need to be aware of the fact that it communicates with the engine instead of with the user directly.

6.2 Fuzzy Matching

With all the data stored in conventional databases, and speed of access remaining an important issue, much of the pattern matching should be done by the databases themselves. It was found that the major problem of a simple fuzzy matching system implemented in Prolog by Ishizuka and Kani [Ishizuka and Kani] was that of speed.

For a practical system, the fuzzy matching facility needs to be an add-on to a host database and to be interfaced to it in an efficient way. Tasks are distributed according to the strength of the software systems involved. Straightforward matching and simple calculations on high volumes of data are carried out in the database, more complex manipulation of smaller volume of data in a specialised system.

6.3 Missing Information

There are two issues of missing information in the database: a whole concept of the domain model may be missing from the database (the database does not contain information about operators of wells), or an individual piece of information may be missing in the database (the total depth of well 259c is unknown). The system deals with both these cases.

An unavailable concept cannot be used in database queries. Users can request the concept to be shown in domain descriptions, but during query generation the concept is de-activated.

A missing value in the database requires more complex treatment. During query processing the system matches the given constraints against values in the database. If during this process it comes across a missing, i.e. unknown, value it can either reject the match to return only those results that definitely match the query, or it can accept the match to return all results that are potential matches. Both approaches are valid and which is the most appropriate depends on the situation. Additionally, if there are missing values in requested information, this must be conveyed to the user in an approapriate way.

In this system the user can control which matching approach is applied and how unavailable information is displayed.

6.4 Mappings

It is important to be able to identify objects reliably in the database. The object oriented approach of query specification and display of information described in this paper relies on the system's ability to use a brief identification of an object as a reference for accessing further information about the object. For this purpose each object is identified by a set of database columns. In many cases this identification is simple (a well's key is its unique number), but it may be more complex involving more than one table.

Another problem is that the information about an object may be spread across different database tables due to database normalisation or differences in the structures of the domain model and the database model. For connecting the information each object has a list of database tables that contain information about it, and specifications of how the tables are joined for the object. Instead of joining each pair of tables in the list it is sufficient to join each table to the object identification described above.

The two issues above are separated from the rest of query processing, so that the query generation process can assume that objects can be identified uniquely and it can pretend that the information about the object resides in a single table. Which data is required for identification and the joins required for the particular query can be picked up from the object specification.

7 Conclusions

A prototype system that embodies the features described above has been implemented and has been demonstrated using three different databases.

This system is under active development, with work planned on a number of aspects. In particular, it is not obvious how best to present a constraint set to the user during initial query formulation—e.g. as an and/or tree, or some kind of object-oriented or form-based display—and various alternatives will be explored in the near future. Similarly, the experience of building the different domain and database models has shown the importance of providing specialised facilities for handling particular aspects of data. In particular, any collections of data include historical information. There are various techniques which are appropriate to manipulating and presenting such data—e.g. presenting a single attribute by graphing its variation over time, or providing "snapshots" of corresponding past values of a number of attributes. Similarly, spatial attributes can naturally be presented using maps, and queried using relationships such as adjacent, near or overlapping. Extending the system to appropriately manipulate temporal and spatial aspects of data will greatly simplify the formulation of domain and database descriptions.

Finally, the fact that most of the system operates without reference to a specific database means that it is not tied to any particular database system or query language. Moreover, it allows the possibility of extending the system to transparently handle multiple databases. This is straightforward as long as each request for information is directed to a single database. Ideally, though, the system should be able to fetch and combine information from any number of databases. To allow this, provisions should be tagged with information regarding the quality of the information the database contains, and the system will have to be able to reason about, and interact with the user concerning, the provenance of information.

In its current state the system succeeds in presenting an object-oriented view of relational database which is intuitively obvious and easy to work with. Query generation is fast, even where fuzzy constraints are involved, and overall performance is acceptable, although post-processing results becomes a bottleneck where queries generate a large number of matches. Configuring the system to handle new databases and domains has proved reasonably straightforward. Indeed, one of the three database/domain specifications was done off-site by a database specialist unconnected with the development team, working entirely from documentation. Nevertheless, specific KB building tools have now been developed to further simplify the process. We are now actively seeking other databases to work with which will extend the library of viewers available and increase the system's functionality.

References

[Chung and Inder] Chung, P.W.H. and Inder, R.
Handling Uncertainty in Accessing Petroleum Exploration Data In *Proceedings of EuroCAIPEP 1991*, Paris, France. October 1991. Also appears, in extended form, in of the *Revue de L'Institut Francais de Petrole*, Vol 47, No. 3, May/June 1992. Also available as AIAI-TR-104.

[Gaines] Gaines, B.R.
Foundations of Fuzzy Reasoning Int. J. Man-Machines Studies, vol 8, pp 623-668, 1976.

[Inder and Stader] Inder, R. and Stader, J.
Bags and Viewers: A metaphor for Intelligent Database Access Technical Report, AIAI-TR-127, 1993.

[Inder and Wells] Inder, R. and Wells, B.
PEXES*: Combining Knowledge and Data in a Tool for the Explorationist* Proceedings of *1991 Conference on Artificial Intelligence in Petroleum Exploration and Production (CAIPEP '91)*, College Station, Texas, 1991, pp 99-106. Also available as AIAI-TR-93.

[Ishizuka and Kani] Mitsuru Ishizuka and Naoki Kani
Prolog-ELF Incorporating Fuzzy Logic New Generation Computing, vol 3, pp 479-486, 1985.

[Schmucker] Schmucker, K.J.
Fuzzy Sets, Natural Language Computations, and Risk Analysis Computer Science Press, 1984.