Natural Language Access to Databases: Interpreting Update Requests[1]

Jim Davidson and S. Jerrold Kaplan
Computer Science Department
Stanford University
Stanford, California 94305

Abstract

For natural language database systems to operate effectively in practical domains, they must have the capabilities required by real applications. One such capability is understanding and performing update requests. The processing of natural language updates raises problems not encountered in the processing of queries. These difficulties stem from the fact that the user will naturally phrase requests with respect to his conception of the domain, which may be a considerable simplification of the actual underlying database structure. Updates which are meaningful and unambiguous from the user's standpoint may not translate into reasonable changes to the underlying database. Update requests may be *impossible* (cannot be performed in any way), *ambiguous* (can be performed in several ways), or *pathological* (can be performed only in ways which cause undesirable side effects).

Drawing on work in Linguistics and Philosophy of Language, we have developed a domain-transparent approach to identifying and performing "reasonable" changes in response to a user's update request, using only knowledge sources typically present in existing database systems. A simple notion of "user model" and explanation with respect to the user's state of knowledge are central to the design. This paper describes a system PIQUE (*Program for Interpretation of Query/Update in English*), which implements this approach.

30 SEPTEMBER 1981

[1]This work is part of the *Knowledge Base Management Systems* project (DARPA contract #N00039-80-G-0132), which is investigating the application of artificial intelligence techniques to issues in the field of database management. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of DARPA or the U.S. Government.

1. Introduction

Natural language is a desirable access mechanism for database systems because it frees the user from the task of understanding the details of the database structure. A number of systems have provided natural language *query* capabilities (e.g., (Sacerdoti, 1977)); however, few of these allow the user to perform *updates* (changes) to the database using natural language. (For an example of one that does allow simple updates, see (Henisz-Dostert & Thompson, 1974).)

The provision of update capabilities introduces problems not seen in handling queries. These problems arise because the user is phrasing his requests with respect to his *view* of the database, which may be a simplification or transformation of the actual database structure. While a well formed query expressed in terms of the user's view of the database will always result in the same answer, regardless of how the query may be mapped into the actual database structure for execution, this is not the case for an update expressed on a view.

Since updates request modification of the content of the database, different mappings of the update request into the actual database structure may result in different effects. Some of these effects may be undesirable or unanticipated. Specifically, the user may make requests that are *impossible* (cannot be performed in any way, due to hidden constraints on the database), *ambiguous* (can be performed in several ways), or *pathological* (can be performed only in ways which cause unanticipated side effects). While human speakers would intuitively reject these unusual readings, a computer program may be unable to distinguish them from more appropriate ones.

For example, a simple request to "Change the teacher of CS345 from Smith to Jones" might be carried out by altering the number of a course that Jones already teaches to be CS345, by changing Smith's name to be Jones, or by modifying a "teaches" link in the database. While all of these may literally carry out the update, they may implicitly cause unanticipated changes such as altering Jones' salary to be Smith's.

Our approach to this problem is to treat updates as requesting that the database be put into a self-consistent state in which the request is satisfied; the problem is then to select the most desirable of (potentially) several such states. The most desirable such state is considered to be the "nearest" one to the current state (in the sense that it involves the least disruption). A set of domain-independent heuristics is used to rank the potential changes along these dimensions.

This process may be guided by various linguistic considerations, such as the difference between "transparent" and "opaque" readings of the user's request, the distinction between the "sense" and "reference" of referring expressions, and the interpretation of counterfactual conditionals.

This paper describes a system *PIQUE* which implements this approach by retaining a model of the user's view, and considering possible methods of performing the update in light of the model. Given an update request, the system generates the set of possible changes to the underlying database that will literally fulfill the request. These candidate changes are then evaluated as to their effects on the user's view, the underlying database, and the database constraints. If possible, an appropriate one is selected; otherwise an informative message is presented to the user.

2. The Problem

As a hypothetical example of the problems which can arise during updates, consider a relational database of employees, salaries, departments, and managers, consisting of two relations:

EMP	SAL	DEPT
Adams	30	Invntry
White	35	Mkting
Brown	25	Sales
Smith	30	Sales
Pullum	25	Sales

DEPT	MGR
Sales	Jones
Mkting	Baker
Invntry	Fisher

and the following dialogue (comments are enclosed in [brackets]):

```
Q1: List the employees and their managers.

R1:     EMP     MGR
        ------------
        Adams   Fisher
        White   Baker
        Brown   Jones
        Smith   Jones
        Pullum  Jones

Q2: Change Brown's manager from Jones to Baker.

R2: Done.
```

[The system has (apparently) fulfilled the user's request]

```
Q3: What is the average salary paid to Jones' employees?

R3: $0.

Q4: List Jones's employees.

R4: NIL
```

[From these responses, the user realizes that something has gone wrong.]

```
Q5: List the employees and their managers.
```

R5:

EMP	MGR
Adams	Fisher
White	Baker
Brown	Baker
Smith	Baker
Pullum	Baker

[The user sees that the system has made two unanticipated changes--changing Smith's and Pullum's managers--in addition to the one that was requested.]

From the user's point of view, his request is meaningful and unambiguous. He sees a set of values, and asks to change one of them. (He might not even know that employees and managers are linked via their departments.)

The problem lies in the fact that his update request can be performed in two ways:

(a) by making the manager of the Sales department be Baker.

(b) by moving Brown from the Sales department to the Marketing department;

Both of these literally fulfill the request. The system, lacking any means for deciding between these, has apparently chosen (a), making Baker the manager of the Sales department, with the unanticipated effect that two other employees have had their managers changed.

3. A More Formal Characterization

This problem can be explained somewhat more formally. Given a database structure, define the user's *view function* F as the transformation that is applied to the database to yield the conceptualization with which the user works. For instance, in the example in section (2), the view function, as defined by Q1, is a transformation consisting of a join and a projection, which is applied to the original two files to yield a single new file with only two attributes. Define the user's *view* as the result of applying the view function to a given state of the database; in the example, this produces a file with five entries, as shown in R1.

A user's update request (call it **u**) is a request to update the *view*. In the example, the request is stated in Q2. Since the view is only 'virtual' (derived from the data), we cannot modify it directly, but must make changes to the underlying database. Call the result of translating the update request to the database level, **T(u)**. The object is to find the change to the underlying database which comes closest to having the desired effect on the user's view. That is, we want the translation **T(u)** which produces a revised database such that, when the view function is applied to that database, the result is the view requested by the user.

In graphical terms:

D represents the initial state of the database, **D'** the state that results after applying the translated update **T(u)**;

$$\begin{tikzcd}
F(D) \arrow[r, "u"] & u(F(D)) \overset{?}{=} F(D') \\
D \arrow[u, "F"] \arrow[r, "T(u)"'] & D' = T(u)(D) \arrow[u, "F"']
\end{tikzcd}$$

In mathematical terms, the mapping **F** from the underlying database **D** to the user's view **F(D)** induces a *homomorphism*. Loosely defined, a homomorphism is a function that preserves the structure of its arguments under given operations. In this case, the operations are changes to the underlying database, and corresponding changes to the user's view. The difficulties with updates expressed on the view (rather than the underlying database) arise from the characteristics of the inverse of this homomorphism: elements in the user's view (states of the "conceptual" database) map under F^{-1} into a set of states of the underlying database. This set may be empty (if the view update cannot be accomplished in any way), or have many elements (in the case of a request which is ambiguous wih respect to **D**). If the mapping **F** is invertible, i.e. F^{-1} is also a function, then an *isomorphism* is induced. In this case, each requested update will have a single, unambiguous interpretation in the underlying database, and the difficulties addessed here do not arise. However, this is not in general the case.

The ideal update translation will produce a state of the database which, when transformed by the user's view function, exactly yields the revised state that he requested. In actuality, our implementation will consider changes to the database that literally fulfill the user's request but may not yield precisely the intended view **u(F(D))**. In the example, there were two translations of the user's request; update (b) yielded the exact view, update (a) a different one.

4. Description of the PIQUE System

We have implemented a system (*PIQUE*) that addresses this problem, by processing update requests in four phases.

(1) decide what the user's current view of the database is;

The system maintains an ongoing model of the user's conception of the database, derived from the dialogue.

(2) use the view to generate a set of *candidate updates* T(u), which perform the update;

When an update comes in, it is assumed to be an update to the user's view. That is, the user requests changes with respect to his conceptualization of the database. The candidate translations are updates to the database, each of which literally accomplishes the user's request.

(3) use a set of ordering heuristics to rank these candidates, in terms of how accurately they fulfill the user's request;

These candidates are evaluated according to the ordering heuristics, to measure how much impact they have on the user's view. For example, a candidate that causes side effects (unrequested changes to the user's view) is ranked lower than one that does not cause such side effects. "Pragmatic" information contained in the database schema is also used in making the decision.

(4) take action, depending on the number of candidates and their ranking;

When the candidates have been ranked, action is taken. This might consist of performing one of the candidates, offering a choice to the user, or explaining why the update cannot be performed at all.

These phases are considered in turn.

4.1. Inferring the user's view

The user of a natural language database system typically has a conception of the database which is a subset of the relations, attributes, connections, and records actually present. In order to interpret updates correctly, the system must take into account the user's current conception of the database. Our approach is to build a user model based on the concepts of which the user has indicated an awareness--those which have occurred in his queries and updates.

This is implemented by making use of the *connection graphs* corresponding to the user's inputs. A system which processes natural language inputs must find paths through the database, defined by operations suc' as *joins*, which connect the concepts mentioned in the input. (The LADDER system, for example provides this service with the help of navigation information stored in a separate *structural schema*.) This set of paths is called the connection graph.

The importance of this work is that the connection graph provides a good model for the structure of the user's view. That is, each query implicitly induces a view of the database which the user holds, at least until the next input. When an update is received, it can be checked for compatibility with the current view, to see if it could be an attempt to update that view. This compatibility test basically checks to see whether the concepts and relationships mentioned in the update are completely contained in the view. (The actual matching criterion is more complicated than simple inclusion, but this will serve for explanatory purposes.) If the update and view are compatible, the user is assumed to be continuing an interaction with that view.

Consider the example of section 2. The user poses a query, which mentions employees and their managers. He then makes an update request of a similar form. Because the update request is compatible with the view induced by the previous query, the user is assumed to be referring to that view, and to be asking to change it. Note that, although *departments* are needed in the connection graph, they are not mentioned by the user, therefore do not appear in the view.

Queries need not always define new views. Under certain circumstances, a query may be a refinement or expansion of a previous view. Consider the sequence "List all the ships in the Mediterranean." "Who are their captains?". The second query merely expands the view defined by the first, by introducing a new attribute.

Views are *stacked* as the dialogue progresses, and updates can be checked for compatibility with all previous views (most recent first). This enables the system to correctly handle a situation in which a user returns to a previous view for further work.

Note that an update also induces a connection graph, just as a query does. If an update request is not compatible with *any* of the views defined previously, the connection graph for the update itself can be used to define the view. This occurs if the user is making an update unrelated to any of the information that he has examined. (For example, if he has a hardcopy, or thinks he knows the contents of the database.) In this case, the view must be inferred from the update alone. Thus, to return to the example of section 2, "Change Brown's manager from Jones to Baker" might be meaningful even if the user has not previously asked about these things.

This strategy is conservative, in that the only concepts that will appear in views are those of which the user has indicated at least some awareness. As a result, the system will never assume a view that is more complex than the one actually held by the user, and thus will never mislead him by introducing a new concept during a response or explanation. The errors which occur will consist of underestimating the user's familiarity with the database; the system will tend to be pedantic, rather than mysterious.

Only minimal cost is required to identify and record the user's view, since navigational work to build the connection graph is required anyway. The testing of views and updates for compatibility is also a simple operation.

This strategy also provides a notion of *focus*: as the user discusses different parts of the database, the view changes automatically. This is important, because the notion of *side effect* changes as the user's focus changes. Changes occurring to previous views are less important than changes occurring to the current view.

The concept of user modelling is well known in artificial intelligence (Mann et al., 1977). A common approach is to record an explicit list of the things the user knows (Appelt, 1980; Cohen, 1978). Our model, however, is much simpler. Given the role of the view information in the inferencing heuristics, this model is adequate for our purposes.

4.2. Generating Candidate Updates

One of the crucial steps of the algorithm described above is the generation of *candidate updates* that can then be evaluated for plausibility. In most cases, an infinite number of changes to the database are possible that would literally carry out the request (mainly by creating and inserting "dummy" values and links). However, this process can be simplified by generating only candidate updates that can be directly derived from the user's phrasing of the request. This limitation is justified by observing that most reasonable updates correspond to different readings of expressions in *referentially opaque* contexts.

A referentially opaque context is one in which two expressions that refer to the same real world concept cannot be interchanged in the context without changing the meaning of the utterance [Quine, 1971]. Natural language database updates often contain opaque contexts.

For example, consider that a particular individual (in a suitable database) may be referred to as "Dr. Smith", "the instructor of CS100", "the youngest assistant professor", or "the occupant of Rm. 424". While each of these expressions may identify the same database record (i.e. they have the same *extension*), they suggest different methods for locating that record (their *intensions* differ). In the context of a database query, where the goal is to unambiguously specify the response set (extension), the method by which they are accessed (the intension) does not normally affect the response (for a counterexample, however, see [Nash-Webber, 1976]). Updates, on the other hand, are often sensitive to the substitution of extensionally equivalent referring expressions. "Change the instructor of CS100 to Dr. Jones." may not be equivalent to "Change the youngest assistant professor to Dr. Jones." or "Change Dr. Smith to Dr. Jones." Each of these may imply different updates to the underlying database.

For operating with an expression in an opaque context, therefore, we must consider the *sense* of the expression, in addition to its *referent* (Frege, 1952). In a database system, this sense is embodied in the procedure used to evaluate the referring expression; the referent is the entity obtained via this evaluation. A request for a *change* to a referring expression is thus not specifically a request to perform a substitution on the referent of the expression, but rather a request to change the database so that the sense of the expression n / has a new referent. That is, after the update, evaluating the same procedure should yield the new (requested) result.

For example, consider a database of ships, ports, and docks, where ships are associated with docks, and docks with ports. Assume that there is currently a ship named Totor in dock 12 in Naples (and no other ship in Naples), and consider the following updates:

> Change *Totor* to Pequod.
> Change *the ship in dock 12* to Pequod.
> Change *the ship in Naples* to Pequod.

The referring expressions (italicized) have the same referent in all three cases, but the senses differ. The expression "Totor" is resolved via a lookup in the *ships* relation; "the ship in dock 12" requires a join between the *ships* and *docks* relations; "the ship in Naples" requires a join between all three relations.

Consider the ways of performing each request, as indicated by the sense of the referring expression. The first version can be implemented only by making a direct substitution on the *ships* relation, corresponding to renaming the ship. The second admits this possibility, but also the possibility of moving a new ship into the dock (if there is already a ship named Pequod). The third allows the above two, plus the possibility of moving a different dock into Naples (if there is a dock somewhere else with Pequod in it). (This will later be ruled out for other reasons, as explained in the next section, but cannot be excluded on purely linguistic grounds.)

Thus, the particular referring expression selected by the user motivates a set of possible actions that may be appropriately taken, but does not directly indicate which is intended or preferred.

This characteristic of natural language updates suggests that the generation of candidate updates can be performed as a *language driven inference* [Kaplan, 1978] without severely limiting the class of updates to be examined. "Language driven inference" is a style of natural language processing in which the inferencing process is driven (and hence limited) by the phrasing of the user's request.

In this instance, the candidate updates are generated by examining the referring expression presented in the update request. The procedure implied by this expression follows an "access path" through the database structure. The candidate updates computed by the program consist of changing links or pointers along that path, or substituting values in the final record(s) identified.

For example, consider the structure of the "ships" database:

```
---------   (a)   ---------   (b)   ---------
| PORTS |  ---->  | DOCKS |  ---->  | SHIPS |
---------         ---------         ---------
```

The candidate translations for the third request (changing "the ship in Naples") correspond to the following changes to the database:

(1) making a change to the SHIPS file (i.e., renaming the ship);
(2) changing link (b) (moving a new ship into the dock);
(3) changing link (a) (moving a new dock into the port).

If the expression "the ship in dock 12" were used, only options 1 and 2 would be generated; similarly, if "Totor" were used, only option 1 would be generated.

4.3. The selection of appropriate updates

At first examination, it would seem to be necessary to incorporate a semantic model of the domain to select an appropriate update from the candidate updates. While this approach would surely be effective, the overhead required to encode, store, and process this knowledge for each individual database may be prohibitive in practical applications. In general, the required information might not be available. What is needed is a general set of heuristics that will select an appropriate update in a reasonable majority of cases, without specific knowledge of the domain.

The heuristics that are applied to rank the candidate updates are based on the idea that the most appropriate one is likely to cause the minimum disruption to the user's conception of the database. This concept is developed formally in the work of Lewis, presented in his book *Counterfactuals* [Lewis, 1973]. In this work, Lewis examines the meaning and formal representation of such statements as "If kangaroos had no tails, they would topple over." (P.8) He argues that to evaluate the correctness of this statement (and similar counterfactual conditionals) it is necessary to construct in one's mind the possible world minimally different from the real world that could potentially contain the conditional (the "nearest" consistent world). He points out that this hypothetical world does not differ only in that kangaroos don't have tails, but also reflects other changes required to make that world plausible. Thus he rejects the idea that in the hypothetical world kangaroos might use crutches (as not being minimally different), or that they might leave the same tracks is the sand (as being inconsistent).

The application of this work to processing natural language database updates is to regard each transaction as presenting a "counterfactual" state of the world, and request that the "nearest" reasonable world in which the counterfactual is true be brought about. For example, the request "Change the teacher of CS345 from Smith to Jones." might correspond to the counterfactual "If Jones taught CS345 instead of Smith, how would the database be different?" along with a speech act requesting that the database be put in this new state.

To select this nearest world, three sources of information are used:

(a) the *side effects* entailed by the different candidates
(b) *pragmatic* information contained in the database schema
(c) *semantic constraints* attached to the database schema

(a) Side effects

As illustrated in the example of section 2, updates may have effects on the user's view and the database beyond those literally requested. Using the rationale of "minimal disruption", updates which do not have side effects are preferable to those that do. For each candidate, we consider the number and type of side effects caused, and rank the candidates accordingly. In data processing terms, the update with the fewest side effects on the user's data sub-model is selected as the most appropriate.

Considering the example from section 2, note that the two candidates have different effects on the user's view. The one which was actually performed--candidate (a), changing the name of the manager of the Sales department--also changes two other values in the view. The other candidate--(b), moving Brown to the Marketing department--does not have these effects. Therefore, the latter more exactly fulfills the user's request, and would be preferred.

The side effects that actually occur for a particular candidate are in a sense accidental, in that they depend on the particular state of the database. For example, the number of side effects caused by changing the manager of the Sales department depends upon how many other employees happen to work in that department. To avoid this artificial property of contingency, a more stable approach is to consider what side effects could result from performing the given candidate in *any* state of the database. This set of *potential* side effects can be determined by analyzing the restrictions in the database schema concerning the cardinality and dependency of relationships between entities. The significance of this concept is that the constraints on cardinality and dependency may be strong enough to ensure that the set of potential side effects (and hence the set of actual ones) is empty--indicating that the given candidate does not have any side effects in the current state, and more importantly, could not have side effects in *any* state.

Consider once again the example of section 2. Of the two updates, (a) causes actual side effects, (b) doesn't. A stronger reason for preferring (b) is that it *cannot* cause side effects, regardless of the state of the database. To see this, note that the cardinality of the relationship between employees and departments is typically N:1--each employee works for only one department. Thus, an employee can have only one manager, and moving the employee to a new department cannot cause any changes to this aspect of the view beyond the one requested. The potential side effects of (a) consist of changes to the managers of employees other than Brown; the two actual side effects are an example of this.

Simple graph algorithms are applied to the database schema to determine which candidates have no potential side effects, and for the others, what side effects they may have. These can be computed more easily than the actual side effects, which must be derived "extensionally" by observing the effects of candidates on the view.

In our ranking of candidates for appropriateness, only potential side effects are considered. Explanations, when needed, are phrased with respect to actual side effects, if any exist, else potential ones.

(b) Pragmatic Information

There may be information in the database schema to help the selection among candidate updates.

For example, certain attributes and links in the schema may be designated at design time as *static*, indicating that they rarely change, or *dynamic*, indicating that they frequently change. This information is used during implementation, to select methods for accessing the information. It may also be of use when ranking candidate updates.

Considering the last example from section 4.2, we note that one of the candidates changes the ship by moving a new dock into Naples. This is consistent within the database, and fulfills the update request; but, the database schema would indicate that such a change is unlikely, and this candidate's desireability would be downgraded. Similarly, there may be general rules that the *names* of things change less often than other attributes.

Note that this information is merely heuristic; if the only candidate is one that involves such a change, it will be performed.

(c) Semantic Constraints

The schema will often contain *semantic constraints* that restrict the allowable states of the database. Examples of these are *functional dependencies* (e.g., "Two employees cannot have the same employee number."), *range constraints* ("No employee can make more than $45K."), and *existence constraints* ("If an employee works in a particular department, there must be a record for that department in the *departments* relation.").

These figure in the process of update interpretation, to rule out candidates which are otherwise acceptable. In the example of section 4.2, if there is already a ship named Pequod in the database, the renaming change could cause a name conflict, resulting in the rejection of this candidate.

Whereas the pragmatic information discussed above was *heuristic*, the semantic constraints are *absolute*. Candidates which violate semantic constraints will never be performed. However, it is still advantageous to generate and consider these candidates, since it is often possible to formulate a meaningful *explanation* for the user about the nonfulfillment of the request.

Our current ordering heuristics incorporate these sources of information. In increasing order of preference, they are:

-updates which violate semantic constraints associated with the database;

-updates which violate pragmatic guidelines;

-updates with side effects on the user's current view;

-updates with no side effects;

While this approach can certainly fail in cases where complex domain semantics rule out the "simplest" change, in the majority of cases it is sufficient to select a reasonable update from among the various possibilities.

Consider again Lewis' "Counterfactual" framework. We see that the restrictions on candidate generation discussed in section 4.2 define the *accessibility* of different states of the world (database); the semantic constraints define *consistency*; pragmatic constraints and side effect information are measures of *distance* between states of the database.

4.4. Action Taken

If one candidate is better than the others, it is performed. If there are a number of candidates which cannot be distinguished by the heuristic ranking, the user is told about them, and offered a choice. If no candidate is admissible (because, for instance, all candidates violate *semantic constraints* on the database), the user is told of this.

In a number of cases, circumstances must be *explained* to the user. For instance, if the candidate actually performed has side effects, the user must be notified of this. If a semantic constraint is violated, the user must be told how.

Our approach to explanation assumes that the user is familiar only with his own view of the database, and so all explanations must be phrased with respect to this understanding (following (McKeown, 1979)). Therefore, options are presented in terms of their "effects" on the user's view (rather than the actual changes proposed), and violations of semantic constraints are discussed with respect to attributes that the user has already seen. In this way, we ensure that explanations are always comprehensible.

5. Examples of the system in operation

PIQUE runs in INTERLISP (Teitelman, 1978), on the PDP-10 at SRI, as part of the *KBMS* system (Wiederhold, 1980). The natural language parser is written in *LIFER*, a semantic grammar system designed by Gary Hendrix (Hendrix, 1977). The database access is done using *SODA*, a relational calculus database query language developed by Bob Moore (Moore, 1979). The SODA interpreter used was written by Bil Lewis, and has been modified and extended by Jim Davidson to handle updates.

Note that some of the information printed by the current system is presented merely for pedagogical purposes, to show the intermediate stages of the computation. In the course of a "real" run, such information (shown indented in the transcripts below) would be suppressed. Additional commentary is enclosed in brackets("[]"). The user's input is preceded by >.

Assume a sample database containing the following information:

Individual employees, with salary, department, and employee number (employee number and name are assumed to be unique):

EMP	SAL	DEPT
Adams	30	Invntry
White	35	Mkting
Brown	25	Sales
Smith	30	Sales
Pullum	25	Sales

EMP	EMPNO
Adams	103
White	431
Brown	554
Smith	222
Pullum	181

Departments have managers and location, and are grouped into divisions (department names are assumed to be unique):

DEPT	MGR	LOC	DIV
Sales	Jones	SF	I
Mkting	Baker	LA	II
Invntry	Fisher	SD	I
Advert	Larkin	NY	II

Divisions are assigned vice presidents:

DIV	VP
I	Lasker
II	Kline
II	Hannan
III	Burgess

1. Example of an update performed using side effect heuristics

Consider once again the example shown in section I, this time with the real system:

```
>(LIST THE EMPLOYEES AND THEIR MANAGERS)

EMP       MGR
--------------------
ADAMS     FISHER
WHITE     BAKER
BROWN     JONES
SMITH     JONES
PULLUM    JONES
```

[Next, the user enters a request to update his view.]

```
Enter next command:

>(CHANGE BROWN'S MANAGER FROM JONES TO BAKER)
```

[The program now generates the candidate updates. As indicated, there are two of these.]

```
      The possible ways of performing the update:

      1.   In the relation DMLD change the MGR attr of the tuple

      DEPT    MGR     LOC     DIV
      ---------------------------
      SALES   JONES   SF      I

      to the value BAKER

      2.   In the relation ESD change the DEPT attr of the tuple

      EMP     SAL     DEPT
      ----------------------
      BROWN   25      SALES

      to the value MKTING
```

[Note that the update of changing the DEPT attribute of the tuple (MKTING BAKER) to Sales would make Baker one of the managers of Brown. But, it would also leave Jones as a manager of Brown, and thus does not fulfill the user's request, even literally. For this reason, it has not even been considered by the system.]

[These candidates are then evaluated, in terms of the accuracy with which they fulfill the user's request. In the absence of any strong preference for either one, the decision is made on the basis of potential side effects to the user's view.]

```
      These translations have the following effects:

      1. Effects are:
      In the view: potentially changing the MGR of other EMPS.
```

```
        2. Effects are:
        None.
```

[The program concludes that update (2) is superior to (1), since (1) has the possibility of changing the manager of other employees. (In actuality, (1) would change the managers of both Smith and Pullum.)]

```
        Desired translation is: 2.

        Revised view is:

        EMP       MGR
        --------------------
        ADAMS     FISHER
        WHITE     BAKER
        BROWN     BAKER
        SMITH     JONES
        PULLUM    JONES
```

[The result accords with the user's wishes; only the requested change has been made to his view.]

2. Example of an update performed using pragmatic information

Consider the same database, with a slightly different dialogue:

```
>(WHAT ARE THE LOCATIONS OF THE EMPLOYEES)

EMP       LOC
--------------------
ADAMS     SD
WHITE     LA
BROWN     SF
SMITH     SF
PULLUM    SF

Enter next command:

>(MOVE ADAMS FROM SD TO LA)
```

[The system interprets this as a request to change Adams' location.]

```
        The possible ways of performing the update:

        1.   In the relation DMLD change the LOC attr of the tuple

        DEPT       MGR      LOC     DIV
        ------------------------------
        INVNTRY    FISHER   SD      I

        to the value LA

        2.   In the relation ESD change the DEPT attr of the tuple
```

```
EMP         SAL       DEPT
-----------------------
ADAMS       30        INVNTRY.

to the value MKTING
```

[Two candidates are identified, corresponding to (1) physically moving the department to a different location, or (2) reassigning the employee.]

[Now, the candidates are evaluated.]

```
These translations have the following side effects on the view:

1. Effects are:
Violation of pragmatic constraints.

2. Effects are:
None.
```

[The "location" attribute of the DMDL relation, representing the location of the department, is marked in the database schema as "static", indicating that it rarely changes. Thus, update (1) is unlikely. The system detects this. Note that update (1) also has potential side effects on the user's view, but the violation of the pragmatic constraint is a stronger reason for rejection.]

```
Desired translation is: 2.

Revised view is:

EMP         LOC
--------------------
ADAMS       LA
WHITE       LA
BROWN       SF
SMITH       SF
PULLUM      SF
```

[Once again, the user's request has been fulfilled exactly.]

3. Example of a request blocked by a semantic constraint:

Assume a *semantic constraint*--a functional dependency stating that each employee number may be assigned to only one employee.

```
>(LIST THE NAMES AND EMPLOYEE NUMBERS FOR ALL EMPLOYEES IN THE
  SALES DEPARTMENT)

EMP      EMPNO
------------
```

```
BROWN   554
SMITH   222
PULLUM  181

>(CHANGE SMITH'S EMPLOYEE NUMBER TO 103)
```

[The program generates the ways of performing the update. There is only one of these.]

```
      The possible ways of performing the update:

      1.   In the relation EN change the EMPNO attr of the tuple

      EMP      EMPNO
      --------------------
      SMITH    222

      to the value 103.
```

[The effects engendered by this candidate are now listed; the candidate would violate a semantic database constraint.]

```
      These translations have the following effects:

      1. Effects are:
      Violation of semantic constraints.
```

[The system now tells the user what has happened, explaining why the update couldn't be performed, and how the semantic constraint would be violated.]

```
This update could not be performed, because of semantic constraints:

The EMPNO value of 103 has already been assigned to the tuple

EMP      EMPNO
------------
ADAMS    103

which has the DEPT value of INVNTRY.
This update would violate the functional dependency EMPNO -> EMP.
```

[Note that, without the DEPT value printed out, the user may not realize why he cannot see the (ADAMS 103) tuple. The explanation is thus phrased with respect to the user view.]

4. Example of a genuinely ambiguous update.

Now, a dialogue concerning a different part of the database:

```
>(WHICH VP'S ARE IN CHARGE OF WHICH DEPARTMENTS?)

VP       DEPT
------------
LASKER   SALES
LASKER  INVNTRY
KLINE   MKTING
KLINE   ADVERT
HANNAN  MKTING
HANNAN  ADVERT

Enter next command:

>(REPLACE LASKER WITH KLINE AS VP IN CHARGE OF THE SALES DEPT)
```

[The system now generates the list of candidate updates. There are two of these.]

```
      The possible ways of performing the update:

      1.   In the relation DV change the VP attr of the tuple

      DIV       VP
      --------------------
      I         LASKER

      to the value KLINE

      2.   In the relation DMLD change the DIV attr of the tuple

      DEPT     MGR      LOC      DIV
      ---------------------------
      SALES    JONES    SF       I

      to the value II
```

[Again, the effects of each on the user's view are computed.]

```
      These translations have the following effects:

      1. Effects are:
      In the view: potentially changing the VP of other DEPTs.

      2. Effects are:
      In the view: potentially inserting or deleting other VPs for this DEPT
```

[Thus, BOTH candidates have side effects on the view. Since we cannot decide *a priori* that one of these is superior to the other, we cannot make a decision here. The only solution is to ask the user. Note that, since the user is presumed to know nothing about the structure of the underlying database, the only meaningful way to distinguish between the updates is to describe them in terms of their (actual) side effects on his view. This is another example of explanation phrased with respect to a view.]

```
There are 2 methods of performing this update.

Update (1) will have the side effect of
replacing the tuple (LASKER INVNTRY) with (KLINE INVNTRY)

Update (2) will have the side effect of
inserting the tuple ((HANNAN SALES))
```

```
Which would you prefer?
>
```

[If the user cannot make a choice, the update is abandoned.]

[Note that the actual side effects are in fact examples of the classes described by the potential ones.]

6. Conclusion

We have presented the salient features of *PIQUE*, a program that performs updates expressed in natural language. Drawing on work in Linguistics and Philosophy of Language, the program implements a domain-transparent approach to identifying and performing "reasonable" changes in response to a user's update request, using only knowledge sources typically present in existing database systems. A simple notion of "user model" and explanation with respect to the user's state of knowledge are central to the design.

The philosophy adopted in the design of PIQUE is somewhat different from that of typical AI systems. Rather than try to capture, represent, and encode the domain- and world-knowledge required to perform a thorough semantic analysis of the problem, we attempt to exploit whatever knowledge is already implicitly or explicitly present in the application (in this case, the content and structure of the database and the user's phrasing of the update request). Consequently, the implementation is simplified and the techniques are more easily transported to new domains.

Of course, the performance of the system suffers when limited information is present. In part because of its generality, there is a definite risk that the system will take inappropriate actions or fail to notice preferable options. A more knowledge-based approach would likely yield more accurate and sophisticated results. The process of responding appropriately to updates could be improved by taking advantage of domain specific knowledge external to the database, using partial case-structure semantics, or tracking dialog focus, to name a few.

To mitigate these shortcomings, the system is engineered to fail "softly", by presenting options to the user or requesting clarifications (by a re-phrasing of the request). As databases encode richer semantic knowledge, as in the proposals of (Wiederhold and El-Masri, 1979; Hammer and McLeod, 1978), the ranking heuristics can be easily extended to take advantage of these additional knowledge sources.

7. Bibliography

Appelt, Douglas E.: "A Planner for Reasoning About Knowledge and Action"; Proc First Nat'l Conf. on AI, 1980, pp 131-133.

Cohen, Philip: "On Knowing What to Say: Planning Speech Acts"; TR #118, CS Dept, University of Toronto, 1978.

Dayal, Umeshwar: "Schema-Mapping Problems in Database Systems"; TR 11-79, Center for Research in Computing Technology, Harvard University, 1979.

Frege, Gottlob: "On Sense and Reference"; trans. Max Black, in Translations from the Philosophical

Writings of Gottlob Frege, P. Geach and M. Black, eds., Blackwell, Oxford, 1952

Hammer, Michael, and Dennis McLeod: "The Semantic Data Model: A Modelling Mechanism for Data Base Applications"; ACM SIGMOD Conference Proceedings, 1978, pp. 26-36.

Hendrix, G.: "Human Engineering for Applied Natural Language Processing"; Proc IJCAI5, 1977, 183-191.

Henisz-Dostert, Bozena, and Frederick B. Thompson: "The REL System and REL English"; in Computation and Mathematical English, A. Zampolli and N. Calzolari, eds., Casa Editrice Olschki, Firenze, 1974.

Kaplan, S. Jerrold, and Jim Davidson: "Interpreting Natural Language Database Updates", in proceedings of the 19th Annual Meeting, Association for Computational Linguistics, Stanford, CA, June, 1981.

Kaplan, S. Jerrold: "Cooperative Responses from a Portable Natural Language Data Base Query System"; HPP-79-19, Computer Science Department, Stanford University, 1979.

Kaplan, S. Jerrold: "Indirect Responses to Loaded Questions", Proceedings of the Second Workshop on Theoretical Issues in Natural Language Processing, Urbana-Champaign, Ill., July, 1978.

Lewis, D.: "Counterfactuals"; Harvard University Press, Cambridge, Ma., 1973.

Mann, William C., James A. Moore, and James A. Levin: "A Comprehension Model for Human Dialogue"; Proc IJCAI5, 1977, pp 77-87.

McKeown, Kathleen R.: "Paraphrasing Using Given and New Information in a Question-Answer System"; Proc. 17th Annual Meeting, Ass'n for Computation Linguistics, 1979, pp 67-72.

Mo[illegible] R.: "Handling Complex Queries in a Distributed Data Base"; TN-170, AI Center, SRI International, October, 1979.

Nash-Webber, B.: "Semantic Interpretation Revisited", BBN report #3335, Bolt, Baranek, and Newman, Cambridge, Ma., 1976.

Quine, .W.V.O.: "Reference and Modality"; in Reference and Modality, Leonard Linsky, Ed., Oxford, Oxford University Press, 1971.

Sacerdoti, Earl D.: "Language Access to Distributed Data with Error Recovery"; Proc IJCAI5, 1977, 196-202.

Teitelman, W.: "Interlisp Reference Manual"; Xerox PARC, Palo Alto, 1978.

Wiederhold, G. and R. El-Masri: "The Structural Model for Database Design"; Proceedings of the International Conference on Entity- Relationship Approach to Systems Analysis and Design, North Holland Press, December 1979, pp 247-267.

Wiederhold, G., S. J. Kaplan, D. Sagalowicz: "Research in Knowledge Base Management Systems"; ACM SIGMOD Record, Vol. 11, No. 3, April, 1981.

Table of Contents