Extending SQL with Customizable Soft Selection Conditions

Gloria Bordogna

CNR Istituto per la Dinamica dei Processi Ambientali
Sezione di Milano – Gruppo Georisorse
via Pasubio 5, Dalmine 24044 (BG), Italy
Tel: +390356224262
e-mail: gloria.bordogna@idpa.cnr.it

Giuseppe Psaila

Facoltà di Ingegneria
Università di Bergamo
Viale Marconi 5, Dalmine 24044 (BG), Italy
Tel: +39 035 2052 355
e-mail: psaila@unibg.it

ABSTRACT

Users of information systems are aware that databases can be a mine of useful information, and would like to express flexible queries over the data possibly retrieving "not perfect" items when the perfect ones, those exactly matching the selection conditions, are not available. Most commercial DBMSs are still based on the SQL for querying. Thus providing some flexibility to SQL can help users to improve their interaction with the systems without requing them to learn a completely novel language. In our approach we allow vague selection conditions based on linguistic predicates, i.e. soft conditions. This topic has been considered in previous works, specifically in SQL/f which is a proposal for extending SQL with soft conditions; however, we think that SQL/f does not completely solve the problem mainly because it does not provide any practical means to customize the meanings of soft conditions to fit specific application domains. Based on these considerations, we propose an extension of SQL which supports customizable soft selection conditions which admit degrees of satisfaction; customizable soft conditions can be defined by users for their specific needs and application domains by means of an SQL like operator. Specifically, this paper proposes an extension of the basic SQL SELECT operator for specifying soft conditions; introduces a new operator for customizing the semantics of the linguistic predicates, provides the formal semantics for the proposed extension of selection conditions.

Categories and Subject Descriptors

H.2.3 Languages (D.3.2) Data description languages (DDL), Data manipulation languages (DML), Query languages

General Terms

Languages.

Keywords.

Flexible query languages; fuzzy set theory; SQL...

1. INTRODUCTION

Users of information systems often have hints or vague ideas of the kind of information that can be hidden in the database but do not have adequate tools for interacting with the DBMSs. The unsatisfying state of the art of database query languages can be exemplified by the common situation in which user requests repeatedly receive empty or no reasonable answer although the user has tried several times to refine his/her selection conditions. This is mainly due to the fact that traditional query languages, such as SQL, force the user to express crisp selection conditions discriminating the retrieved items into two disjoint sets: the retrieved and the rejected items. For example, looking for cities that are big implies the translation of the linguistic predicate big into a crisp condition such as city.population greater than 1 Million inhabitants”. Consequently, a city whose population is just 999K is not retrieved, while it is intuitive that it is anyway big. Furthermore, the retrieved items are undifferentiated while it is natural to think that a city with a population of 2 Million inhabitants satisfies the information need to a better extent than a city of 1.1 million inhabitants.

A well known paradigm to make a query language flexible is to allow the specification of preferences so as to shift from „exact matching“ towards „a best possible matching“. Preferences can be modeled by soft selection conditions expressed by linguistic predicates that can be interpreted as soft constraints, such as the requirement for something is big, or city is close to another one. [2][8][9]. Then, the retrieved items can be ranked in decreasing order of the soft condition satisfaction value. Specifically SQL/f [9] is the proposal for extending SQL with soft conditions in the fuzzy set framework.

In our approach we retain this interpretation and formal framework and extend it so as to make it possible the customization of the semantics of the linguistic predicates to fit specific application domains and user needs. We think that, in order to make the language really flexible and useful from a practical point of view, this feature must be available to the user. Often the semantics of linguistic predicates cannot be univocally defined, but several coexistent interpretations can be useful in the...
same application context. Let us recall as an example the semantics of close on the spatial domain: requiring that two cities be close one another implies a very different notion of closeness that when requiring that two building blocks are close within a city. Closeness is then a notion that depends on the considered scale.

Based on these considerations, we propose an extension of SQL which supports customizable soft selection conditions with the following characteristics:

- Linguistic predicates can be embedded in an extended SQL query for specifying soft selection conditions admitting degrees of satisfaction. This degree of satisfaction computed for a tuple can be used for ranking the items reflecting this way a notion of relevance to the user preference.
- Linguistic predicates can be customized by users to fit their specific needs and application domains.
- Linguistic predicates can be defined with multiple coexistent semantics and can be specified in queries so as to fit their right meaning.

Specifically, this paper proposes an extension of the basic SQL SELECT operator; introduces a new operator for defining linguistic predicates, and provides the formal semantics for the proposed extension of selection conditions.

The paper is organised as follows. Section 2 analyses the related literature on flexible querying in databases and briefly describes SQL/f; Section 3 introduces the proposed extensions to SQL; Section 4 defines the formal semantics. Finally, Section 5 draws the conclusions.

2. FLEXIBLE QUERYING IN DATABASES

The problem of defining a flexible query languages for databases has been considered in previous works [1],[2],[6],[7],[8],[9],[10]. As Kiesling pointed out [2] there is an “exact world” where user wishes can be satisfied completely or not at all. Database queries in this context are characterized by crisp constraints that deliver exactly the ideal desired items, if these are stored in the database. On the other side there is also the “real world” where users are leaning to express preferences for items having certain characteristics. In this context preferences can be interpreted as wishes: there is no guarantee that these wishes are satisfied by the retrieved items, but the user is prepared to accept “not perfect” items.

In cooperative database systems the technique of query relaxation have been studied in order to cope with the problem of the empty set resulting from the evaluation of crisp query conditions [3],[4]. In some approaches the notion of preferences has been interpreted as that of priorities between selection conditions [2],[5],[6]. On the line of this interpretation are the approaches in [2],[6] in which the preferences are specified directly by using binary preference relations.

We assume a different interpretation that is derived by the approaches defined within fuzzy databases in which preferences are interpreted as soft conditions on attribute values [1],[8],[9],[10], i.e., tolerant conditions admitting degrees of satisfaction so that there is a gradual transition from acceptable and excluded items. A soft condition is expressed by means of a linguistic predicate represented by a fuzzy set [8],[11],[12]. The first introduction of preferences has been done for the SQL and the relational database model [12],[9],[10]. A natural extension of a relation r, defined as a subset of D=D1 D2 ... Dn, is provided by the notion of fuzzy relation rf, defined as a fuzzy subset of D, i.e., each tuple d of rf is associated with a membership degree rf(d) in [0,1]. rf(d) is interpreted as the degree of satisfaction of the vague predicates in the query. In order to satisfy the closure property, a regular relation can be seen as a kind of fuzzy relation with all the tuples having the same membership degree equal to 1. Fuzzy SQL (SQL/f) has been defined by extending the relational algebra [1],[8], so as to operate on fuzzy relations. For their formal definition refer to [8]. The basic SQL/f query allows to specify soft conditions in the WHERE clause. An example of SQL/f query on the relation CITIES:= (id, name, population, location) is:

```
SELECT 5 0.6 C.name
FROM CITIES as C
WHERE C.population IS big
```

in which the soft condition is expressed by “IS big”. It imposes the evaluation of the degree of satisfaction of the linguistic predicate “IS big” by the values of the attribute C.population of the relation CITIES. The intermediate fuzzy relation resulting from the evaluation of the soft condition is projected on the attribute C.name and the best 5 tuples over the threshold 0.6 are returned to the user. Notice that, in SQL/f queries, linguistic values such as big are specified independently of the context to which they are applied. This can create problems since semantics of linguistic predicates might depend on the context. Further, in SQL/f users cannot explicitly define the semantics of the linguistic predicates since no SQL/f operator is provided to this purpose. This is a severe limitation of SQL/f that makes it inadequate from a practical point of view.

3. THE PROPOSED EXTENSIONS TO SQL

In order to support customizable soft conditions we had in mind the following design guidelines:

- the semantics of the soft selection conditions must be formalized to make clear how the soft selection conditions interact with other clauses in the SQL SELECT operator.
- Soft selection conditions should be easily customizable; in other words, what is the meaning of “something is big”? It depends on the specific application context, i.e. the actual context of “something”, thus the semantics of linguistic predicates must be customizable.
- The closure property of the SQL SELECT operator must be strictly preserved as it is; in other words, a SELECT statement takes standard tables as input and generates a standard table as output; no attributes with special implicit meaning (such as membership degree) should be present in tables.

The result is both an extension of the basic SQL SELECT operator and a new SQL-like operator, named CREATE TERM SET, for defining the semantics, i.e., the meaning, of linguistic predicates and the term sets of linguistic terms that can be made available to query the database.

Like in SQL/f, soft conditions are specified through linguistic predicates identifying fuzzy subsets of the attribute domains and are specified in the WHERE clause of the extended SQL query, thus producing a fuzzy relation as a result of their evaluation. However, the overall SELECT operator is extended, in order to make all clauses coherent with this new feature. Linguistic predicates exploited in the soft condition are defined by means of
the CREATE TERM SET operator; by means of this operator users are allowed to define the term sets of the linguistic values of attributes and their semantics. In this way, the same linguistic value can be defined with a distinct meaning depending on the reference attribute. In the following sub-sections, we introduce both operators.

### 3.1 The SELECT operator

The evaluation of linguistic predicates requires that tuples under selection have associated a membership degree. In our proposal, the membership degree is a property of each tuple, while it is not an attribute as in SQL. It can be obtained by means of a special function DEGREE(), allowed only in the select clause. Since the closure property must be strictly maintained, we extend the FROM clause to allow the specification of an attribute (for each table in the clause) to be interpreted as the membership degree of the tuples.

**Syntax.** The revised syntax for the SELECT operator is the following.

```sql
SELECT [TOP n] result_schema
FROM from_clause_spec
WHERE soft_selection_condition
[ DEGREE THRESHOLD dt ]
[ GROUP BY group_attr_list
[ HAVING group_selection_condition ] ]
[ ORDER BY sort_list ]
FROM Clause. The syntax of the FROM clause is substantially unchanged w.r.t. standard SQL. The only difference concerns table specifications, that become as follows:

table [AS table_alias] [WITH expr AS DEGREE]
```

W.r.t. standard SQL, we allow an additional optional WITH expr AS DEGREE sub-clause: if not specified, the membership degree for tuples coming from the specified table is set to 1; if specified, for each tuple the value assumed by the expression expr (based on attributes contained in the specified table) becomes the membership degree of the tuple.

**WHERE Clause.** The WHERE clause presents significant changes w.r.t. the standard; in particular,

- **Soft_selection_condition** is the condition by means of which tuples are selected and ranked; it is now a soft condition, since it can be composed of standard SQL crisp predicates and linguistic predicates, connected by AND, OR and negated by NOT. For the sake of simplicity, we do not consider here the possibility to specify soft aggregation connectives, although possible and coherent as in SQL. Linguistic predicates have the form:

  ```sql
tuple or numerical-expression
  IS linguistic-value
  IN Term-set
  where a tuple is a tuple of attributes, while linguistic-value is defined in the specified Term-set. The numerical-expression is a usual SQL numerical expression.
  For example, the linguistic predicate (sales, expected_sales) IS ‘weak’ IN SalesFactor
  in which sales and expected_sales are numerical attributes, is equivalent to
  sales / expected_sales IS ‘weak’ IN SalesFactor
  ```

For each tuple, the evaluation of the condition assigns a new membership degree to the tuple, based both on the initial membership degree and on the evaluation of the degree of satisfaction of the linguistic predicates.

The optional sub-clause DEGREE THRESHOLD allows to specify a filtering threshold over the membership degree of each tuple: if present, only tuples with membership degree greater than `dt` are selected, otherwise only tuples with membership degree greater than 0 are selected. `dt` is greater than 0 and less than 1 (see Section 4 for a complete definition of semantics).

**GROUP BY clause.** The GROUP BY clause behaves similarly to standard SQL, but each group has a membership degree as well, which is obtained as the minimum membership values of tuples belonging to the same group. The HAVING clause is based on the standard SQL syntax and semantics; thus, group_selection_condition allows standard aggregate functions and predicates over aggregate functions and grouping attributes; in this context we do not allow the specification of linguistic quantifiers aggregations over the grouping attributes.

**SELECT clause.** As in the standard SQL SELECT operator, the attribute list appearing in the SELECT clause (i.e. result_schema) defines the schema for the table generated by the SELECT statement; here, we extend SQL by allowing the use of function DEGREE(), whose value is the membership degree of tuples (if the GROUP BY clause is not present) or groups (if the GROUP BY clause is present).

**ORDER BY clause.** As in the standard SELECT operator, the ORDER BY clause sorts the tuples in the result table; We allow to specify the DEGREE() function as sort key. The TOP sub-clause in the SELECT clause inserts into the result table only the first `n` sorted tuples.

### 3.2 Examples

Suppose we are working on a relational database containing the table `US_Stores` collecting information about stores of a company; specifically:

```sql
US_Stores(name, state, distance_from_Capital, sales, expected_sales)
```

First of all, we might wish to extract states with at least 3 stores having weak sales. The query might be the following.

```sql
SELECT state, DEGREE AS M
FROM US_Stores
WHERE (sales, expected_sales) IS 'weak'
IN SalesFactor DEGREE THRESHOLD 0.6
GROUP By state
HAVING COUNT(*) > 3
```

First of all, the FROM clause takes table `US_Stores`; the membership degree for all tuples in the table is, by default, set to 1. Then, the WHERE clause evaluates the soft condition on every tuple in the `US_Stores` table; the membership degree is obtained by evaluating the linguistic predicate, that exploits the term sets named `SalesFactor`. The membership of each tuple `t` is then the satisfaction degree of the soft condition `weak` obtained by evaluating $\mu_{weak}(t, sales / t.expected_sales)$ on the sales attribute value. At this point, as specified by the DEGREE_THRESHOLD sub-clause, only tuples with membership degree greater than 0.6 are selected and subsequently grouped by attribute `state`; then
only states with more than 3 stores with weak sales are inserted into the result table. Observe that the state name is associated with the membership degree \( M \), that must be necessarily greater than 0.6. Suppose now that the previous query is used to look for stores in critical states with weak sales and located far away from the state capital, which are the best 10 top tuples.

\[
\text{SELECT TOP 10 name}
\text{FROM Critical_states as CS}
\text{INNER JOIN US_Stores on}
\text{CS.state = US_Stores.state}
\text{WHERE distance_from_Capital IS}
\text{‘close’}
\text{IN CityDistances AND}
\text{(sales, expected_sales) IS}
\text{‘weak’}
\text{IN SalesFactor}
\text{ORDER BY DEGREE() DESC}
\]

To answer this query, it is necessary to join table Critical_State with table US_Stores; in accordance with the definition of the fuzzy Cartesian product the membership degree obtained in the previous query, i.e., the \( M \) attribute of table Critical_States, is combined with the membership degree obtained by evaluating the soft condition, and the minimum value is taken as final membership degree. Observe that the soft condition exploits another term set, named CityDistances.

After the evaluation of the WHERE clause, tuples are ordered in descending order w.r.t. their membership degree, then only the first 10 tuples with the highest membership degree are inserted into the result table.

### 3.3 An Operator for Defining Term Sets and Linguistic Values

To make the query language adaptable to user needs, we define a new SQL like operator for defining term sets (and the meanings of linguistic values), which can be used in the WHERE clause to specify linguistic predicates by means of the \( \text{IS} .. \text{IN} \) operator. This operator is named \text{CREATE TERM-SET}. This way, the semantics of linguistic predicates can be easily customized.

\[
\text{CREATE TERM-SET name}
\text{[ NORMALIZED WITHIN (min, max) ]}
\text{[ EVALUATE expression}
\text{WITH PARAMS List-of-params ]}
\text{VALUES list of Linguistic-Value-Definition}
\]

in which \( name \) is the name of a term set.

The set of linguistic values in the term set is defined by means of the \( \text{VALUES} \) clause. Each \text{Linguistic-Value-Definition} is a pair (linguistic value, meaning):

- **linguistic value**, is a string identifying a linguistic value; ex., 'very far'.
- **meaning**: is a 4-tuple of ordered values in the range \([0,1]\) defining the trapezoidal membership function (x coordinates of the left bottom/left top/right top/right bottom corners) of the fuzzy set identified by linguistic value. The domain of the membership function is normalized within the range \([0,1]\). The possibly missing \text{NORMALIZED} keyword specifies if the domain of the numerical value evaluated by the \( \text{IS} .. \text{IN} \), i.e., the domain of the membership functions associated with the linguistic value must be normalized w.r.t. the range \([0,1]\); values less than (greater than) \( \text{min} \) (\( \text{max} \)) are always evaluated as \( \text{min} \) (\( \text{max} \)). If the \text{NORMALIZED} clause is not specified, values are always evaluated w.r.t. the range \([0,1]\).

The same term set can be evaluated by the \( \text{IS} .. \text{IN} \) operator over different data types. This is allowed by the non-empty list of \text{EVALUATE} clauses: each occurrence defines a specific function to be computed, whose values are “compared” with the linguistic values. This way, depending on the data type of the left operand in the \( \text{IS} .. \text{IN} \) operator, the proper function is applied. Note that the \text{WITH PARAMS} sub-clause defines the list of formal parameters appearing in the expression. Consider the term set CityDistances to evaluate soft distance conditions.

\[
\text{CREATE TERM-SET CityDistances}
\text{NORMALIZED WITHIN (0, 1000)}
\text{EVALUATE DIST WITH PARAMS DIST AS FLOAT}
\text{VALUES (‘very far’, (0.8, 0.9, 1.1)),}
\text{(‘far’, (0.5, 0.75, 1,1)),}
\text{(‘medium_distance’, (0.1,0.45,0.55,0.7)),}
\text{(‘close’, (0,0, 0.025, 0.1)),}
\text{(‘very close’, (0,0,0.01,0.02))}
\]

Notice that distances are normalized within 0 Km and 1000 Km as specified by the \text{NORMALIZED WITHIN} parameters. This way, the term set CityDistances can be exploited to express conditions on different numerical domains of distances. The numerical value is directly matched over the membership function. Similarly, we can define the term set SalesFactors, that characterizes sales w.r.t. the expected sales. Two terms are defined: ‘weak’ (for weak sales) and ‘Strong’. Note that it is possible to exploit the term set to evaluate either a pre-computed sales factor (first \text{EVALUATE} clause) or a pair of values, where the former value is the amount of effective sales, and the second value is the expected amount of sales (see the second \text{EVALUATE} clause, where the ratio is computed from two parameters).

\[
\text{CREATE TERM-SET SalesFactor}
\text{EVALUATE sales_factor WITH}
\text{PARAMS sales_factor AS float}
\text{EVALUATE sales/expected WITH}
\text{PARAMS sales AS float,}
\text{expected AS float}
\text{VALUES (‘weak’, (0, 0, 0.3,0.6)),}
\text{(‘strong’, (0.3, 0.6, 1,1))}
\]

### 4 FORMAL SEMANTICS

#### Membership Function after \text{INNER JOIN}

Consider an \text{INNER JOIN} operation between two tables \( T_1 \) and \( T_2 \), and two tuples \( t_1 \in T_1 \) and \( t_2 \in T_2 \) having membership degrees \( \mu(t_1) \) and \( \mu(t_2) \), respectively. If \( t_1 \) and \( t_2 \) can be joint, the joint tuple \( t = t_1 \bullet t_2 \) has membership degree \( \mu(t) = \text{Min}(\mu(t_1), \mu(t_2)) \) since it is interpreted as the fuzzy Cartesian product of \( T_1 \) and \( T_2 \). [9]

#### Membership Function of Linguistic Values

We assume that, in most applications, linguistic values are a means to express a tolerance to under-satisfaction of a crisp condition. In specifying the semantic of a linguistic value the objective is then to define the tolerance range to under-satisfaction. In this perspective the shape of the membership function \( \mu_{lv} \) of a linguistic value \( lv \) can
be simply and adequately defined by a trapezoidal function, \((lb, lt, rt, rb)\), where \((lt-lb)\) and \((rb-rt)\) are the tolerance ranges.

If \(0 \leq e \leq lb\) then 0

\[
\mu_t(e) = \begin{cases} 
0 & \text{if } lb < e < lt \text{ then } (e-lb)/(1 + (1/(lt-lb))) \\
\text{Max}(\mu_t(sub-cond'),(t)) & \text{if } lt \leq e \leq rt \text{ then } 1 \\
\text{Min}(\mu_t(sub-cond'),(t)) & \text{if } rt < e \leq rb \text{ then } (e-rt)/(1 - (1/(rb-rt))) \\
1 & \text{if } rb \leq e \leq 1 \text{ then } 0
\end{cases}
\]

The **IS . . IN** Operator. A predicate based on the **IS . . IN** operator has the form:

\( t \text{ IS lv IN TS} \)

where \(t\) is a tuple of objects, \(lv\) is a linguistic value in the term set \(TS\), i.e., \(lv \in TS\). If the type of \(t\) matches an expression \(exp\), defined for \(TS\), the membership function for the operator application is defined as

\[
\mu_{t \text{ IS lv IN TS}}(t) = \mu_{lv}(exp)
\]

Recall that the membership values generated by crisp operators are only 0 and 1.

The **WHERE** Clause. We assume that each tuple \(t\) in a source crisp relation \(R\) has a membership degree \(\mu(R)(t)=1\), as defined by the FROM clause. The **WHERE** clause specifies a soft condition \(\varphi\), which assigns to tuple \(t\) a new membership degree. Applying associative properties, \(\varphi\) can be seen either as \(\varphi = sub-cond' \lor sub-cond''\), where \(\lor\) is a logical operator **AND**, **OR**, or as a negated condition \(\varphi = \neg(sub-cond')\), where \(sub-cond'\) and \(sub-cond''\) can be either composed conditions or simple predicates. In accordance with the usual definitions [9] the semantics of the three logical operators is:

- If \(\varphi = sub-cond' \land sub-cond''\), then \(\mu_{t}(t)=\text{Min}(\mu_{sub-cond'},(t))\)
- If \(\varphi = sub-cond' \lor sub-cond''\), then \(\mu_{t}(t)=\text{Max}(\mu_{sub-cond'},(t))\)
- If \(\varphi = \neg(sub-cond')\), then \(\mu_{t}(t)=1 - \mu_{sub-cond'}(t)\)

Then, the membership degree of a tuple \(t\) after the evaluation of \(\varphi\) is \(\mu(t) = \text{Min}(\mu(R)(t), \mu_{t}(t))\) [9]. Finally, given a membership threshold \(dt\), the **WHERE** clause selects only tuples \(t\) such that \(\mu(t) > dt\).

Membership after **GROUP BY**. Consider a group of tuples \(g\) obtained by grouping tuples as specified in the **GROUP BY** clause. The membership degree for the overall group is

\[
\mu(g) = \text{Min}(\mu(t_1), ..., \mu(t_i), ..., \mu(t_n))
\]

where \(g = \{ t_1, ..., t_i, ..., t_n \} \).

5 CONCLUSIONS

We proposed a feasible solution towards the definition of a flexible database query language, that allows the expression of customizable soft conditions.

We retain the basic idea of SQL/f [8,9] to extend standard SQL for supporting soft conditions. However, our proposal improves flexibility by allowing to specify the context of the linguistic predicates by means of a new comparison operator, named **IS . . IN**, in the **WHERE** clause of the **SELECT** operator. The semantics of the linguistic predicates can be easily defined with respect to a specific context by means of the **CREATE TERM-SET** operator, defined in this paper as well.

6 REFERENCES


