The handling of select-project-join operations in a relational framework supported by possibilistic logic

Guy de Tré, Jörg Verstraete, Axel Hallez, Tom Matthé, Rita de Caluwe
Database, Document and Content Management
Department of Telecommunications and Information Processing
Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Guy.DeTre@UGent.be

Abstract

Traditionally, flexible querying systems are supported by a logical framework that is based on satisfaction degrees. Alternatively, certainty about query satisfaction can also be modelled by possibility and necessity measures. This approach was extended by our group in order to handle the cases of missing information more adequately. In this paper, it is presented how our extended logical framework can be used to support flexible querying on conventional relational databases. More specifically, the focus of the paper is on the handling of selection–projection–join queries. Motivated examples are provided to illustrate the used concepts.

Keywords: flexible querying, conventional relational databases, select–project–join operations.

1 Introduction

Since many years, an emphasis has been put on research that aims to make database systems more flexible and better accessible. An important aspect of flexibility is the ability to deal with imperfections of information, like imprecision, vagueness, uncertainty or incompleteness. Imperfection of information can be dealt with at the level of the data modelling, the level of database querying or both.

The key idea in flexible querying is to introduce preferences inside database queries [3]. This can be done at two levels: inside elementary query conditions and between query conditions. Preferences inside query conditions allow to express that some values are more adequate than others, whereas preferences between query conditions are used to associate different levels of importance with the conditions.

To support preferences, query languages like SQL and OQL and their underlying algebraic frameworks have been generalized. Hereby, the possible extensions and flexible counterparts of the algebraic data manipulation operators have been studied [1, 12, 2, 15, 8]. As the main objective of flexible querying is to refine Boolean conditions, which are either completely true or completely false, it is sufficient that the underlying logical framework supports some notion of ‘degree of satisfaction’. Alternatively, an underlying logical framework based on possibility and necessity measures can be used to express certainty about query satisfaction. This approach, as originally presented in [10] does not provide any discussion of inapplicability of information at the logical level, nor offer a presentation of a formal framework for coping with it together with other null values —despite the fact that inapplicability is handled with a special domain value (⊥) in the data model—.

As illustrated in [5, 6], extended possibilistic truth values (EPTVs), which will be further described in the next section, can be used to express (un)certainty about query satisfaction
in flexible database querying: the EPTV representing the extent to which it is (un)certain that a given database record belongs to the result of a flexible query can be obtained by aggregating the calculated EPTV’s that denote the extents to which it is (un)certain that the record satisfies the different criteria imposed by the query. Moreover, the logical framework based on EPTVs extends the approach presented in [10] and explicitly deals with the inapplicability of information during the evaluation of the query conditions: if some part of the query conditions are inapplicable, this will be reflected in the resulting EPTV.

In this paper, we further study the handling of flexible queries on conventional, relational databases supported by a logical framework of EPTVs. More specifically, the focus is on the selection–projection–join operations. The remainder of the paper is organized as follows. In the next Section 2, some preliminaries on EPTVs are described. In Section 3, the use and impact of EPTVs in the querying of conventional, relational database are dealt with. Subsection 3.1 describes the framework and introduces an illustrative database, Subsection 3.2 deals with the selection operation, Subsection 3.3 deals with the projection operation and Subsection 3.4 deals with the join operation. An example of a general query, where all operations are subsequently applied, is given in Subsection 3.5. Finally, some conclusions are given in Section 4.

2 Some preliminaries on EPTVs

The concept ‘extended possibilistic truth value’ (EPTV) is defined as an extension of the concept ‘possibilistic truth value’ (PTV), which was originally introduced in [9]. PTVs provide an epistemological representation of the truth of a proposition, which allows to reflect knowledge about the actual truth. Their semantics is defined in terms of a possibility distribution.

With the understanding that $P$ represents ‘True’ and $F$ represents ‘False’), the so-called possibilistic truth value (PTV) $\tilde{t}(p)$ of a proposition $p \in P$ is formally defined by the mapping

$$\tilde{t}: P \rightarrow \tilde{\phi}(I)$$

which associates a fuzzy set $\tilde{t}(p)$ with each $p \in P$.

The semantics of this associated fuzzy set $\tilde{t}(p)$ is defined in terms of a possibility distribution. With the understanding that

$$t: P \rightarrow I$$

is the mapping function that associates the value $T$ with $p$ if $p$ is true and associates the value $F$ with $p$ otherwise, this means that

$$\forall x \in I ((\pi_t(p))(x) = \mu_{t(p)}(x))$$

i.e., $\forall p \in P (\pi_t(p) = \tilde{t}(p))$ where $\pi_t(p)(x)$ denotes the possibility that the value of $t(p)$ conforms to $x$ and $\mu_{t(p)}(x)$ is the membership grade of $x$ within the fuzzy set $\tilde{t}(p)$.

If the truth value of a proposition $p$ is unknown, modelling is done by using the PTV $\tilde{t}(p) = \{(T,1),(F,1)\}$

which denotes that it is completely possible that $p$ is true ($T$), but it is also completely possible that $p$ is false ($F$).

A PTV $\tilde{t}(p)$ is normalized if at least one of the membership grades $\mu_{\tilde{t}(p)}(T)$ and $\mu_{\tilde{t}(p)}(F)$ is equal to 1. With the possibilistic interpretation, normalization implies that at least one of the Boolean truth values should be completely possible.

Whenever it does not make sense to consider a ‘regular’ truth value, e.g., when some parts of the proposition are not defined or are not applicable, an empty fuzzy set can be used for modelling, representing in this case an inapplicable truth value. However, this approach has the disadvantage of making it impossible to work with normalized possibilistic truth values and moreover does not allow for a gradual representation of inapplicability. As an alternative approach, the Boolean set $I$ can be
extended with an extra element \( \perp \) that represents an ‘undefined’ truth value that will be used to model ‘inapplicability’ [5]. This leads to the concept EPTV.

With the understanding that \( P \) represents the universe of all propositions and \( \tilde{\wp}(I^*) \) denotes the set of all ordinary fuzzy sets in the universe \( I^* = \{ T, F, \perp \} \), the so-called extended possibilistic truth value (EPTV) \( \tilde{t}^*(p) \) of a proposition \( p \in P \) is formally defined by the mapping

\[
\tilde{t}^* : P \rightarrow \tilde{\wp}(I^*)
\]

which associates a fuzzy set \( \tilde{t}^*(p) \) with each \( p \in P \).

The semantics of the associated fuzzy set \( \tilde{t}^*(p) \) are defined in terms of a possibility distribution. With the understanding that

\[
t^* : P \rightarrow I^*
\]

is the mapping function which associates the value \( T \) with \( p \) if \( p \) is true, associates the value \( F \) with \( p \) if \( p \) is false and associates the value \( \perp \) with \( p \) if (some of) the elements of \( p \) are not applicable, undefined or not supplied, which means that

\[
(\forall x \in I^*)(\pi_{t^*(p)}(x) = \mu_{t^*(p)}(x))
\]

i.e., \( (\forall p \in P)(\pi_{t^*(p)} = \tilde{t}^*(p)) \) where \( \pi_{t^*(p)}(x) \) denotes the possibility that the value of \( t^*(p) \) conforms to \( x \) and \( \mu_{t^*(p)}(x) \) is the membership grade of \( x \) within the fuzzy set \( \tilde{t}^*(p) \).

New propositions can be constructed from existing propositions, using logical operators. A unary operator ‘\( ^\wedge \)' is provided for the negation (NOT) of a proposition and binary operators ‘\( \wedge \)', ‘\( \vee \)', ‘\( \Rightarrow \)' and ‘\( \Leftrightarrow \)' are respectively provided for the conjunction (AND), disjunction (OR), implication (IF THEN) and equivalence (IFF) of propositions. The arithmetic rules to calculate the EPTV of a composite proposition and the algebraic properties of EPTVs are presented in [5]. For example, the rules for conjunction can be summarized as

\[
\forall p, q \in P : \tilde{t}^*(p \ \text{AND} \ q) = \tilde{t}^*(p) \wedge \tilde{t}^*(q)
\]

where

\[
\wedge : \tilde{\wp}(I^*) \times \tilde{\wp}(I^*) \rightarrow \tilde{\wp}(I^*) : (\tilde{U}, \tilde{V}) \mapsto \tilde{U} \wedge \tilde{V}
\]

is defined by

\[
\mu_{\tilde{U} \wedge \tilde{V}}(T) = \min(\mu_{\tilde{U}}(T), \mu_{\tilde{V}}(T))
\]

\[
\mu_{\tilde{U} \wedge \tilde{V}}(F) = \begin{pmatrix}
\min(\mu_{\tilde{U}}(T), \mu_{\tilde{V}}(F)), \\
\min(\mu_{\tilde{U}}(F), \mu_{\tilde{V}}(T)), \\
\max(\min(\mu_{\tilde{U}}(F), \mu_{\tilde{V}}(F)), \\
\min(\mu_{\tilde{U}}(T), \mu_{\tilde{V}}(T)), \\
\min(\mu_{\tilde{U}}(\perp), \mu_{\tilde{V}}(\perp))
\end{pmatrix}
\]

\[
\mu_{\tilde{U} \wedge \tilde{V}}(\perp) = \begin{pmatrix}
\min(\mu_{\tilde{U}}(T), \mu_{\tilde{V}}(\perp)), \\
\min(\mu_{\tilde{U}}(\perp), \mu_{\tilde{V}}(T)), \\
\min(\mu_{\tilde{U}}(\perp), \mu_{\tilde{V}}(\perp))
\end{pmatrix}
\]

These rules, and analogously the rules for disjunction and negation, are obtained by applying Zadeh’s extension principle [14] to the operators of the strong three-valued Kleene logic [11]. Kleene logics are truth-functional, which means that according to these systems, the behavior of a logical operator is mirrored in a logical function combining Kleene truth values. Therefore, the extended truth value of every composed proposition can be calculated as a function of the extended truth values of its original propositions.

3 Flexible querying on conventional relational databases

In order to support flexible querying in conventional relational databases, the logical framework based on EPTVs must be extended with some additional facilities. In the next subsection a basic extension for relational database schemas is presented: all relations are extended with an extra attribute (Contains), whose values are EPTVs that express the extent to which the associated tuples belong to the relation. Hereby, it is implicitly assumed that the heading of a relation corresponds to a predicate and all tuples that belong to the relation are propositions that should not evaluate to false, i.e. that have an associated EPTV that differs from \({(F, 1)}\). Additionally, an illustrative sample database is given. The following subsections respectively deal with extensions of the selection, projection and join operations. A more general example is given in the last subsection.
3.1 Framework and illustrative database

In flexible querying, query satisfaction is a matter of degree. In order to express the extents to which tuples belong to the answer set of a query, an extra attribute (Contains) is added to the relation that represents the result set. This attribute has as corresponding domain the set of all EPTVs. Its values are calculated during query processing, as will be explained in the following subsections. In fact, an associated EPTV expresses the certainty about the compatibility of the tuple with the results expected by the user.

In order to guarantee the relational closure property of the set of relational algebra operators [4], all relations in the database are extended with an extra attribute (Contains). If no more information is available, for simplification, it can be assumed that all tuples initially have the value \( \{ (T, 1) \} \) as associated EPTV. The value \( \{ (T, 1) \} \) could, for example, be the default value that is assigned to the tuple on insertion. In a more general case, users might be allowed to assign their own truth values, hereby expressing that the tuple belongs to the relation, only to the given extent.

The relational database used as illustration contains data about artists and their music recordings and consists of two relations, named ‘Artist’ and ‘Recording’. Each tuple in Artist represents information about a music performer and is characterized by a unique artist ID (AID) —which is the primary key attribute—and an age attribute (Age)\(^1\). Each tuple in Recording represents information about a music recording and is characterized by a title (Title), the corresponding artist ID of the performer (AID) —which is a foreign key that refers to relation Artist— and the highest position of the recording in the international hit charts (Top). The primary key of Recording is composed and consists of the attributes Title and AID. Examples of both relations are given in Tables 1 and 2.

\(^1\)In real-world databases, the birthday of the artist, rather than the age, will be stored.

### Table 1: Example of relation Artist.

<table>
<thead>
<tr>
<th>AID</th>
<th>Age</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>36</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>A2</td>
<td>51</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>A3</td>
<td>22</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>A4</td>
<td>23</td>
<td>{ (T, 1) }</td>
</tr>
</tbody>
</table>

### Table 2: Example of relation Recording.

<table>
<thead>
<tr>
<th>Title</th>
<th>AID</th>
<th>Top</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Blessed’</td>
<td>A1</td>
<td>3</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>‘Love’</td>
<td>A3</td>
<td>1</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>‘Heaven’</td>
<td>A4</td>
<td>1</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>‘Love’</td>
<td>A2</td>
<td>1</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>‘Son’</td>
<td>A3</td>
<td>( \bot_{\text{Int}} )</td>
<td>{ (T, 1) }</td>
</tr>
<tr>
<td>‘Stranger’</td>
<td>A2</td>
<td>5</td>
<td>{ (T, 1) }</td>
</tr>
</tbody>
</table>

Each attribute \( A_i \) has an associated domain \( \text{dom}_{A_i} \) of valid values. Each associated domain \( \text{dom}_{A_i} \) contains a domain specific ‘undefined’ element \( \bot_{A_i} \) that is used to model cases where a regular domain value is not applicable (cf. [10]).

3.2 The selection operation

In relational algebra [4], the selection operation, also called the restriction operation, is written in the following general format:

\[
a \text{WHERE} \ e
\]

where \( a \) denotes a database relation and \( e \) is a truth-valued function, also called the restriction condition, whose parameters are some subset of the attributes of \( a \).

The selection operation restricts relation \( a \), by discarding all tuples of \( a \) that do not satisfy \( e \) at all, i.e., that have a calculated truth value that differs from false.

In the proposed extension, the truth-valued function \( e \) is further generalized to a function that evaluates to an EPTV. Examples of such functions are the ‘IS’ function and the generalizations of the comparison operators like ‘=’, ‘\( \neq \)’, ‘<’ and ‘>’. As an illustration, only the definition of the ‘IS’ function is described below. Definitions for the comparison operators are given in [7].
With the understanding that \( a \) is the crisp stored value of attribute \( A \) and \( \mu_L \) is the membership function of the fuzzy set that represents the values desired by the user, the EPTV of the proposition ‘\( A \) IS \( L \)’ is defined by:

\[
\{(T, \mu_T), (F, \mu_F), (\perp, \mu_{\perp})\}
\]

where

\[
\begin{align*}
\mu_T &= \mu_L(a) \\
\mu_F &= \begin{cases} 
1 - \mu_L(a) & \text{if } a \neq \perp_A \\
0 & \text{if } a = \perp_A 
\end{cases} \\
\mu_{\perp} &= \begin{cases} 
1 - \mu_L(\perp_A) & \text{if } a = \perp_A \\
0 & \text{if } a \neq \perp_A 
\end{cases}
\]

The result set is obtained by evaluating \( e \) and by calculating the EPTVs that are associated with the resulting tuples. Hereby, it is important and necessary that the EPTVs of the original relation \( a \) are appropriately dealt with. Therefore, the conjunction operator \( \wedge \), presented in Section 2 is applied to the original EPTV and the EPTV that is obtained from the evaluation of \( e \). Only tuples with a resulting EPTV that differs from \( \{(F, 1)\} \) belong to the resulting relation.

As an example consider the following query:

**Query 1**

**Recording WHERE Top IS High**

This query selects all ‘Recording’-tuples with a high ranking position in the charts. The linguistic term ‘High’ is, e.g., modelled by the fuzzy set \( L = \{(1, 1), (2, 1), (3, 0.8), (4, 0.4)\} \). The tuples belonging to the result set of Query 1 are given in Table 3. For every tuple in this result set, the corresponding EPTV is calculated as the conjunction of the EPTV associated with the tuple in the original relation and the ETPV that is obtained by applying the ‘IS’ function as defined above.

EPTVs allow to model the partial satisfaction of a flexible query condition (tuple ‘Blessed’). It also might be the case that the flexible condition is completely satisfied (tuples ‘Love’ and ‘Heaven’) or not satisfied at all (tuple ‘Stranger’—this tuple is not in the result set of the query—). If some part of the data is not defined, e.g., due to the fact that the recording is not commercially available and therefore could not enter hit charts (tuple ‘Son’), this is explicitly reflected in the associated EPTV.

### 3.3 The projection operation

The algebraic projection operation is written in the following general format [4]:

\[
a\{X, Y, \ldots, Z\}
\]

where \( a \) denotes a database relation and \( X, Y, \ldots, Z \) are regular attributes of \( a \).

The result of the projection operation is a relation with a heading derived from the heading of \( a \) by removing all attributes not mentioned in the set \( \{X, Y, \ldots, Z\} \) and a body consisting of all tuples of \( a \), restricted to the values of attribute \( \{X, Y, \ldots, Z\} \).

In the proposed extension, the extra attribute `Contains` is also added to the heading of the resulting relation. For each tuple \( t \) in the body of the resulting relation, the corresponding EPTV is calculated as the disjunction of all EPTVs that are associated with tuples \( t' \) in \( a \) that have the same attribute values for the attributes in \( \{X, Y, \ldots, Z\} \), i.e.

\[
t(\text{Contains}) = \bigvee_{t'(X,Y,\ldots,Z)=t(X,Y,\ldots,Z)} t'(\text{Contains})
\]

In this way, the relational closure property is guaranteed with respect to the projection operator.

As an example consider the following query:

**Query 2**
Recording \{\text{Title,Top}\}

This query selects all ‘Recording’-tuples, but restricts their tuple values to the values of attributes \textit{Title} and \textit{Top}. The corresponding EPTVs of the original \textit{Recording} relation are copied to the resulting relation. The tuples belonging to the result set of Query 2 are given in Table 4.

<table>
<thead>
<tr>
<th>\text{Title}</th>
<th>\text{Top}</th>
<th>\text{Contains}</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Blessed’</td>
<td>3</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>‘Love’</td>
<td>1</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>‘Heaven’</td>
<td>1</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>‘Son’</td>
<td>\bot_{\text{Int}}</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>‘Stranger’</td>
<td>5</td>
<td>{(T,1)}</td>
</tr>
</tbody>
</table>

Hereby, the EPTV of tuple (‘Love’,1,\{(T,1)\}) is obtained by taking the disjunction of the EPTVs of tuple (‘Love’,A3,1,\{(T,1)\}) and tuple (‘Love’,A2,1,\{(T,1)\}) of \textit{Recording}, i.e., by calculating \{(T,1)\} \lor \{(T,1)\}.

3.4 The join operation

Consider the two relations \(a\) and \(b\) with respective attribute sets

\[{XY\{\text{Contains}_a\}}\text{ and }{YZ\{\text{Contains}_b\}}\]

where

\[\begin{align*}
X &= \{X_1,X_2,\ldots,X_m\}, \\
Y &= \{Y_1,Y_2,\ldots,Y_n\}
\end{align*}\]

and

\[Z = \{Z_1,Z_2,\ldots,Z_p\}.
\]

This means that, the attributes \(Y_1, Y_2, \ldots, Y_n\) are common to the two relations, \(X_1, X_2, \ldots, X_m, \text{Contains}_a\) are the other attributes of \(a\) and \(Z_1, Z_2, \ldots, Z_p, \text{Contains}_b\) are the other attributes of \(b\). \text{Contains}_a\) is the extra attribute for the associated EPTVs in \(a\), whereas \text{Contains}_b\) is the extra attribute for the associated EPTVs in \(b\).

The algebraic (natural) join operation is written in the following format [4]:

\[a \text{ JOIN } b\]

3.5 A general query example

In order to illustrate the impact of the use of EPTVs on full select–project–join queries, a more general example is given below.

Consider the query:

\[\text{Query 4}\]

\[((\text{Recording JOIN Artist})\]

\[\text{WHERE Age IS Young and Top IS High})\}

\{\text{AID,Age}\}

This query searches the database for young, successful artists and returns their artist ID (\text{AID}) and age (\text{Age}). Under the assumptions
that the linguistic term ‘Young’ is modelled by the fuzzy set with membership function:

\[ \mu_{\text{young}}(x) = \begin{cases} 1, & \text{if } x < 30 \\ -x/20 + 5/2, & \text{if } 30 \leq x \leq 50 \\ 0, & \text{if } x > 50 \text{ or } x = \perp_{\text{Int}} \end{cases} \]

and a successful artist is characterized by having at least one recording with a high ranking in the charts, where the linguistic term ‘High’ is again modelled by the fuzzy set \( L = \{(1,1), (2,1), (3,0.8), (4,0.4)\} \), the result of the flexible query can be obtained as follows:

Step 1. First, the join operation is performed. This yields the same result as in Query 3 (cf. Table 5).

Step 2. Next, the restriction conditions are evaluated individually and the resulting EPTVs are combined using the conjunction operator \( \land \), as given in Section 2. These results are summarized in Tables 6 and 7. For simplicity, only the attributes \( \text{Title} \) and \( \text{AID} \) are presented.

The EPTVs resulting from the evaluation of the condition on \( \text{Age} \) are presented in column \( \text{EPTV}_{\text{Age}} \), the EPTVs resulting from the evaluation of the condition on \( \text{Top} \) are presented in column \( \text{EPTV}_{\text{Top}} \), whereas the aggregated EPTVs are given in column \( \text{EPTV}_{\text{Agg}} \).

Step 3. The aggregated EPTVs are aggregated with the EPTVs obtained in step 1. As can be seen in Table 5, all these EPTVs equal \( \{(T,1)\} \), what makes that this aggregation has no effect on the EPTVs in column \( \text{EPTV}_{\text{Agg}} \).

Step 4. Finally, the projection operation is applied and the resulting relation of the query is obtained. This relation is presented in Table 8.

### Table 7: Intermediate results (Continued).

<table>
<thead>
<tr>
<th>Title, AID</th>
<th>( \text{EPTV}_{\text{Agg}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Blessed’, A1</td>
<td>{(T,0.7), (F,0.3)}</td>
</tr>
<tr>
<td>‘Love’, A3</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>‘Heaven’, A4</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>‘Love’, A2</td>
<td>{(F,1)}</td>
</tr>
<tr>
<td>‘Son’, A3</td>
<td>{\perp, 1}</td>
</tr>
<tr>
<td>‘Stranger’, A2</td>
<td>{(F,1)}</td>
</tr>
</tbody>
</table>

### Table 8: Result set of Query 4.

<table>
<thead>
<tr>
<th>AID</th>
<th>Age</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>36</td>
<td>{(T,0.7), (F,0.3)}</td>
</tr>
<tr>
<td>A3</td>
<td>22</td>
<td>{(T,1)}</td>
</tr>
<tr>
<td>A4</td>
<td>23</td>
<td>{(T,1)}</td>
</tr>
</tbody>
</table>

Because, in the intermediate result, there are two tuples with values A3 and 22 for \( \text{AID} \) and \( \text{Age} \) — once with associated EPTV \( \{(T,1)\} \) and once with associated EPTV \( \{\perp, 1\} \) — the resulting EPTV for \( (A3, 22, \{(T,1)\}) \) is obtained by \( \{(T,1)\} \lor \{(\perp, 1)\} = \{(T,1)\} \).

### 4 Conclusion

In this paper, we have presented how a logical framework, based on EPTVs, can be used to support flexible querying on conventional relational databases. More specifically, in this paper, the focus is on the extension of the selection, projection and (natural) join operations.

The presented approach can be implemented in an extra module that runs on top of the query engine of a relational database management system.

The presented approach can be further extended. In order to obtain a full flexible query language, the extension of the other relational algebra operators must be studied. Furthermore, the introduction of weights on query conditions to denote user preferences, can be
considered. Other fields of ongoing research are the use and incorporation of flexible integrity constraints and the generalization of the approach towards ‘fuzzy’ databases, i.e., databases that can contain imperfect (imprecise, vague, incomplete or uncertain) data.

References


