PREFERENCE RULES IN DATABASE QUERYING

Sergio Greco, Cristian Molinaro and Francesco Parisi
DEIS, University of Calabria, 87036 Rende, Italy

Keywords: Deductive databases, prioritized queries, preferences.

Abstract: The paper proposes the use of preferences for querying databases. In expressing queries it is natural to express preferences among tuples belonging to the answer. This can be done in commercial DBMS, for instance, by ordering the tuples in the result. The paper presents a different proposal, based on similar approaches deeply investigated in the artificial intelligence field, where preferences are used to restrict the result of queries posed over databases. In our proposal a query over a database $DB$ is a triple $(q, P, \Phi)$, where $q$ denotes the output relation, $P$ a Datalog program (or an SQL query) used to compute the result and $\Phi$ is a set of preference rules used to introduce preferences on the computed tuples. In our proposal tuples which are “dominated” by other tuples do not belong to the result and cannot be used to infer other tuples. A new stratified semantics is presented where the program $P$ is partitioned into strata and the preference rules associated to each stratum of $P$ are divided into layers; the result of a query is carried out by computing one stratum at time and by applying the preference rules, one layer at time. We show that our technique is sound and that the complexity of computing queries with preference rules is still polynomial.

1 INTRODUCTION

The growing volume of available information poses new challenges to the database and artificial intelligence communities. Recent researches have investigated new techniques in accessing large volumes of data such as user-centered access to information, information filtering and extraction and policies to reduce data presented to users. An interesting direction deeply studied in the artificial intelligence and non-monotonic reasoning fields consists in the use of preferences to express priorities on the alternative scenarios. The paper presents a logical framework wherein preferences are used to restrict the result of queries posed over a database. This is an important aspect in querying large databases such as those used by search engines. In this context, the result of a query contains only tuples which are not dominated by other tuples and dominated tuples cannot be used to infer new information. The novelty of the presented approach is that preferences are stratified and applied one stratum at time. A second innovative aspect of this proposal is that preferences on both base and derived atoms are considered as well as general (recursive) queries which can be expressed by means of stratified Datalog.

Example 1 Consider a database $DB = \{\text{fish, beef}\}$ and a program $P$ consisting of the two rules:

\begin{verbatim}
red-wine ← beef
white-wine ← fish
\end{verbatim}

Assume now to have a query defined by the rules in $P$ and the preference $\rho_1 = \text{red-wine} \succ \text{white-wine} \Leftarrow \text{beef}$ stating that if there is beef, we prefer red-wine to white-wine. The set of preferred atoms contains the base atoms fish and beef and the derived atom red-wine (the atom white-wine is not preferred). Assume now to also have the preference $\rho_2 = \text{fish} \succ \text{beef}$ stating that we prefer fish to beef. In this case, first the preference rule $\rho_2$, and next the preference rule $\rho_1$, are considered. However, $\rho_1$ cannot be applied as beef is not in the preferred set.
of atoms. Consequently, the set of preferred atoms, with respect to the preference rules \( p_2 \) and \( p_1 \), is \{fish,white-wine\}.

**Contributions.** In this paper we study the use of preferences in querying databases. We consider general (stratified) Datalog queries and general preferences: the head of preference rules may contain atoms belonging to different relations and the body consists of a conjunction of literals. A semantics where both query and preferences are partitioned into strata is defined. Under such a semantics, the query is computed one stratum at time and for each stratum (of the query), the preferences are applied one stratum at time.

**Related Work.** The increased interest in preferences in logic programs is reflected by an extensive number of proposals and systems for preference handling. Most of the approaches propose an extension of logic programming by adding preference information. The most common form of preference consists in specifying a strict partial order on rules (Delgrande et al., 2003; Gelfond and Son, 1997; Sakama and Inoue, 2000; Zhang and Foo, 1997), whereas more sophisticated forms of preferences also allow priorities to be specified between conjunctive (disjunctive) knowledge with preconditions (Brewka et al., 2003; Sakama and Inoue, 2000) and numerical penalties for suboptimal options (Brewka, 2004).

Considering the use of preferences in querying databases, an extension of relational calculus expressing preferences for tuples in terms of logical conditions has been proposed in (Lacroix and Lavency, 1987). Preferences requiring non-deterministic choice among atoms which minimize or maximize the value of some attribute has been proposed in (Greco and Zaniolo, 2002). An extension of Datalog with preference relations, subsuming the approach proposed in (Kostler et al., 2003), whereas an extension of SQL including preferences has been proposed in (Kießling, 2002; Kießling and Kostler, 2002). In the last proposal several built-in operators and a formal definition of their combinations (i.e. intersection, union, Pareto composition, etc.) has been considered. Borzsonyi et al. proposed the skyline operator (Borzsonyi et al., 2001), to filter out a set of "interesting" point (i.e. not dominated by any other point) from a potential large set of points. An extension of SQL with a skyline operator has been also proposed. A framework for specifying preferences using logical formulas and its embedding into relational algebra has been introduced in (Chomicki, 2003). The paper also introduces the winnow operator which generalizes the skyline operator. The implementation of winnow and ranking is also studied in (Torlone and Ciaccia, 2002). Algorithms for computing skyline operators are also studied in (Kossmann et al., 2002; Papadakis et al., 2003; Chomicki et al., 2003). In (Agrawal and Wimmers, 2002) the use of quantitative preferences (scoring functions) in queries is proposed.

In this work, in contrast with previous proposals, general preferences and a different (stratified) semantics, which we believe to be more intuitive, are considered.

## 2 BACKGROUND

Familiarity with disjunctive logic programs and disjunctive deductive databases is assumed (Ullman, 1988).

**Datalog Programs.** A term is either a constant or a variable. An atom is of the form \( p(t_1, \ldots, t_k) \), where \( p \) is a predicate symbol of arity \( k \), and \( t_1, \ldots, t_k \) are terms. A literal is either an atom \( A \) or its negation \( \neg A \). A (Datalog) rule \( r \) is a clause of the form

\[
A ← B_1, \ldots, B_n, \neg B_{m+1}, \ldots, \neg B_n, \varphi \quad n \geq 0
\]

where \( A, B_1, \ldots, B_n \) are atoms, whereas \( \varphi \) is a conjunction of built-in atoms of the form \( u \theta v \) where \( u \) and \( v \) are terms and \( \theta \) is a comparison predicate. \( A \) is the head of \( r \) (denoted by \( Head(r) \)), whereas the conjunction \( B_1, \ldots, B_n, \neg B_{m+1}, \ldots, \neg B_n, \varphi \) is the body of \( r \) (denoted by \( Body(r) \)). It is assumed that each rule is safe, i.e. a variable appearing in the head or in a negative literal also appears in a positive body literal. A (Datalog) program is a finite set of rules. A not-free program is called positive. The Herbrand Universe \( \mathcal{U}_P \) of a program \( P \) is the set of all constants appearing in \( P \), and its Herbrand Base \( \mathcal{B}_P \) is the set of all ground atoms constructed from the predicates appearing in \( P \) and the constants from \( \mathcal{U}_P \). A term (resp. an atom, a literal, a rule or a program) is ground if no variable occurs in it. A rule \( r' \) is a ground instance of a rule \( r \) if \( r' \) is obtained from \( r \) by replacing every variable in \( r \) with some constant in \( \mathcal{U}_P \); ground(\( P \)) denotes the set of all ground instances of the rules in \( P \).

An interpretation \( M \) for a Datalog program \( P \) is any subset of \( \mathcal{B}_P \); \( M \) is a model of \( P \) if it satisfies all rules in ground(\( P \)). The (model-theoretic) semantics for positive \( P \) assigns to \( P \) the set of its minimal models \( P \models (\mathcal{M} \models P) \), where a model \( M \) for \( P \) is minimal if no proper subset of \( M \) is a model for \( P \). For any interpretation \( M \), \( M \models P \) is the ground positive program derived from ground(\( P \)) by 1) removing all rules that contain
PREFERENCE RULES IN DATABASE QUERING

a negative literal not A in the body and A ∈ M, and 2) removing all negative literals from the remaining rules. An interpretation M is a stable model of ϕ if and only if M ∈ SM(ϕM) (Gelfond and Lifschitz, 1988). For general ϕ, the stable model semantics assigns to ϕ the set SM(ϕ) of its stable models. It is well-known that stable models are minimal models (i.e. SM(ϕ) ⊆ M(M(ϕ)) and that for negation free programs minimal and stable model semantics coincide (i.e. SM(ϕ) = M(M(ϕ))).

Given a Datalog program ϕ, Gϕ = (Vϕ, Eϕ) denotes the dependency graph associated with ground(ϕ) where Vϕ consists of all ground atoms appearing in ground(ϕ), whereas there is an arc from B to A ∈ Eϕ if there is a rule r in ground(ϕ) such that Head(r) = A and B ∈ Body(r); the arc is said to be marked negatively if B appears negated in the body of r. The dependency graph Gϕ = (V, E) associated with ϕ is built by considering the ground program derived from ϕ by eliminating all terms (i.e. every atom p(t) is replaced by p). A ground atom p(t) depends on a ground atom q(u) if there is a path in Gϕ from q(u) to p(t). Analogously, a predicate symbol p depends on a predicate symbol q if there is a path in Gϕ from q to p. The dependency is negated if there is an arc marked negatively in the path.

A partition π0, ..., πk of the set of all predicate symbols of a Datalog program ϕ, where each πi is called stratum, is a stratification of ϕ if for each rule r in ϕ the predicates that appear only positively in the body of r are in strata lower than or equal to the stratum of the predicate in the head of r, and the predicates that appear negatively are in strata lower than the stratum of the predicate in the head of r. The stratification of the predicates defines a stratification of the rules of ϕ into strata {ϕ1, ..., ϕk} where a stratum ϕi contains rules which define predicates in πi. A Datalog program is called stratified if it has a stratification. Stratified (normal) programs have a unique stable model which coincides with the stratified model, obtained by computing the fixpoints of every stratum in their order.

**Queries.** Predicate symbols are partitioned into two distinct sets: base predicates and derived predicates. Base predicates correspond to database relations defined over a given domain and they do not appear in the head of any rule, whereas derived predicates are defined by means of rules. Given a set of ground atoms D, a predicate symbol p and a stratified program ϕ, D[p] denotes the set of p-tuples in D, while ϕD denotes the program derived from the union of ϕ with the facts in D, i.e. ϕD = ϕ ∪ D. The semantics of ϕD is given by the stratified model (which coincide with the unique stable model) of ϕD. The answer to a query Q = (g, ϕ) over a database D, denoted by Q(D), is given by M[g] where M = SM(ϕD). In the following we also denote with M(D) = SM(ϕD) the application of ϕ to D; therefore Q(D) = ϕ(D)[g].

## 3 PREFERENCE RULES AND QUERIES

This section presents a framework for expressing preferences in the evaluation of queries posed on a given database. The framework is based on the introduction of preference rules, whose syntax is inspired to the management of priorities in the artificial intelligence field, logic programming and database querying (Brewka el al., 2003; Delgrande et al., 2003; Gelfond and Son, 1997; Sakama and Inoue, 2000; Zhang and Foo, 1997).

### 3.1 Syntax

A prioritized program consists of a set of standard rules (Datalog program) and a set of preference rules. As rules expressing preferences eliminate tuples which are derived by means of standard rules (Datalog program) we first introduce a standard stratification of the Datalog program to fix the order in which standard rules are applied. Preference rules are associated to each subprogram (stratum) and applied after the subprogram has been evaluated. Let start by introducing the concept of standard stratification.

**Definition 1** The standard stratification of a stratified program ϕ consists of k strata {ϕ1, ..., ϕk} where k is the minimal value such that for each ϕi and for each pair of predicates p and q defined in ϕi either they are mutually recursive or they are independent (i.e. p does not depend on q and q does not depend on p).

In the following, given an atom p(t), str(p(t)) denotes the stratum of the predicate symbol p (or equivalently of the subprogram in which p is defined) in the standard stratification.

**Definition 2** A preference rule ϕ is of the form:

\[ A ≺ C \leftarrow B_1, ..., B_m, \text{not } B_{m+1}, ..., \text{not } B_n, \varphi \]  

where A, C, B1, ..., Bn are atoms, and ϕ is a conjunction of built-in atoms.

Also in this case we assume that rules are safe. In the above definition A ≺ C is called head of the preference rule (denoted as Head(p)), whereas the conjunction B1, ..., Bm, not B_{m+1}, ..., not Bn is called...
body (denoted as \(\text{Body}(p)\)). Moreover, we denote with \(\text{Head}_1(p)\) and \(\text{Head}_2(p)\) the first and the second atom in the head of \(p\), respectively (i.e. \(\text{Head}_1(p) = A\) and \(\text{Head}_2(p) = C\)).

The intuitive meaning of a ground preference rule \(p\) is that if the body of \(p\) is true, then the atom \(A\) is preferable to \(C\) (we also say that the atom \(C\) is dominated by the atom \(A\)). This means that in the evaluation of a prioritized program \(\langle g, \varphi, \Phi \rangle\) the model defining its semantics cannot contain the atom \(C\) if it contains the atom \(A\) and the body of the preference rule is true.

Let \(\Phi\) be a preference program, i.e. a set of preference rules. The transitive closure of \(\text{ground}(\Phi)\) is \(\Phi_\star = \text{ground}(\Phi) \cup \{\langle A \succ C \iff \text{body}_1, \text{body}_2 \mid \exists A \succ B \iff \text{body}_1 \in \Phi_\star \land \exists B \succ C \iff \text{body}_2 \in \Phi_\star\}\). Analogously, we define \(\Phi_\star^n\) as the closure of the set of ground preference rules derived from \(\Phi\) by replacing every atom \(p(i)\) with \(p\) and deleting built-in atoms.

**Definition 3** A (ground) preference program \(\Phi_g\) is layered if it is possible to partition it into \(n\) layers \(\{\Phi_g[1], \ldots, \Phi_g[n]\}\) as follows:

- For each ground atom \(A\) such that there is no ground rule \(\rho \in \Phi_g\) such that \(\text{Head}_2(\rho) = A\), \(\text{layer}(A) = 0\);
- For every ground atom \(C\) such that there is a rule \(\rho\) of the form (1) (i.e. such that \(\text{Head}_2(\rho) = C\), \(\text{layer}(C) > \max\{\text{layer}(B_1), \ldots, \text{layer}(B_m), 0\}\) and \(\text{layer}(C) \geq \text{layer}(A)\);
- The layer of a preference rule \(\rho \in \Phi_g\) denoted as \(\text{layer}(\rho)\), is equal to \(\text{layer}(\text{Head}_2(\rho))\);
- \(\Phi_g[i]\) consists of all preference rules associated with the layer \(i\).

**Example 2** Consider the set of preference rules \(\Phi\):

\[
\begin{align*}
p_1 & : \text{fish} \succ \text{beef} \iff \text{white} \\
p_2 & : \text{red-wine} \succ \text{white-wine} \iff \text{beef} \\
p_3 & : \text{white-wine} \succ \text{red-wine} \iff \text{fish}
\end{align*}
\]

The transitive closure \(\Phi_\star\) consists of the rules \(p_1, p_2, p_3\) plus the following rules:

\[
\begin{align*}
p_4 & : \text{red-wine} \succ \text{red-wine} \iff \text{beef, fish} \\
p_5 & : \text{white-wine} \succ \text{white-wine} \iff \text{fish, beef}
\end{align*}
\]

\(\Phi_\star\) is partitioned into the two layers \(\Phi_\star[1] = \{p_1\}\) and \(\Phi_\star[2] = \{p_2, p_3, p_4, p_5\}\).

As it will be clear in the next subsection, preference rules of the form \(A \succ A \iff \text{body}\) are useless and can be deleted. Therefore, in the above example \(\Phi_\star[2] = \{p_2, p_3\}\).

**Example 3** Consider the set of preference rules \(\Phi\):

\[
\begin{align*}
p_1 & : \text{fish} \succ \text{beef} \iff \text{white-wine} \\
p_2 & : \text{red-wine} \succ \text{white-wine} \iff \text{beef}
\end{align*}
\]

According to \(p_1\) the layer of \(\text{beef}\) must be greater than the layer of \(\text{white-wine}\), whereas according to \(p_2\) the layer of \(\text{white-wine}\) must be greater than the layer of \(\text{beef}\). Thus, the set of preference rules is not layered. 

Observe that in the above definition, in order to compute the closure of the ground instantiation of \(\Phi\), we need to know the database \(\mathcal{D}\mathcal{B}\) containing all constants in the database domain. Therefore, checking whether \(\Phi_\star\) can be partitioned into layers cannot be done at compile-time. It is possible to define sufficient conditions which guarantee that the set of preference rules can be partitioned into layers by considering the (ground) program \(\Phi_\star\) instead of the program \(\Phi_\star^n\). This means that if \(\Phi_\star\) can be partitioned into layers, the set \(\Phi_\star^n\) can be partitioned into layers as well, although the layers of \(\Phi_\star^n\) may be different from the layers of \(\Phi_\star\) (the layers of \(\Phi_\star^n\) define a “refinement” of the layers of \(\Phi_\star\)).

**Definition 4** A prioritized query is of the form \(\langle g, \varphi, \Phi \rangle\) where \(g\) is a predicate symbol denoting the output relation, \(\varphi\) is a (stratified) Datalog program and \(\Phi\) is a set of preference rules.

As said before, the intuitive meaning of a prioritized query \(\langle q, \varphi, \Phi \rangle\) over a database \(\mathcal{D}\mathcal{B}\) is that the atoms derived from \(\varphi\) and \(\mathcal{D}\mathcal{B}\) must satisfy the preference conditions defined in \(\Phi\).

**Definition 5** A prioritized query \(Q = \langle q, \varphi, \Phi \rangle\) is said to be well formed if \(\Phi_\star\) is layered and for every ground atom \(C\) such that there is a rule \(\rho\) of the form (1) (i.e. such that \(\text{Head}_2(\rho) = C\) it holds that

1. \(\text{str}(C) \geq \max\{\text{str}(A), \text{str}(B_1), \ldots, \text{str}(B_m)\}\), and
2. \(A, B_1, \ldots, B_m\) do not depend on \(C\) in \(\varphi\).

In the following we assume that our queries are well formed. Sufficient conditions can be defined on the base of the dependency graph \(g(\varphi)\).

### 3.2 Semantics

First we analyze the case where \(\Phi\) defines preferences on databases atoms and next we consider the case where \(\Phi\) expresses preferences on base and derived atoms, i.e. also on atoms defined in \(\varphi\).

#### 3.2.1 Preferences On Base Atoms

It is assumed here to have a query \(Q = \langle q, \varphi, \Phi \rangle\) and that the preference rules in \(\Phi\) express preferences only among base atoms. As said before, \(\Phi_\star^n\) can be partitioned into \(n\) layers \(\Phi_\star^n = \{\Phi_\star^n[1], \ldots, \Phi_\star^n[n]\}\).

**Definition 6** Let \(\mathcal{D}\mathcal{B}\) be a set of ground atoms, \(\Phi\) a set of preference rules such that \(\Phi_\star^n = \{\Phi_\star^n[1], \ldots, \Phi_\star^n[n]\}\), and \(\mathfrak{u}, \mathfrak{v}\) two atoms in \(\mathcal{D}\mathcal{B}\). We say...
that \( t \) is preferable to \( u \) with respect to \( \Phi^*_g[i] \) (denotes as \( t \gg u \in \Phi^*_g[i] \)) if

1. \( \exists (t \gg u \in \text{body}_1) \in \Phi^*_g[i] \) s.t. \( DB_1 \models \text{body}_1 \), and
2. \( \exists (u \gg t \in \text{body}_2) \in \Phi^*_g[i] \) s.t. \( DB_2 \models \text{body}_2 \).

The set of tuples in \( DB \) which are preferred with respect to \( \Phi^*_g[i] \) is \( \Phi^*_g[i](DB) = \{ t \mid t \in DB \land \exists u \in DB \ s.t. \ u \gg t \} \).

Observe that \( \Phi^*_g \) could contain preference rules of the form \( A \gg A \in \text{body} \). Such preferences are useless as they are not used to infer preferences among ground atoms and can be deleted from \( \Phi^*_g \).

**Example 4** Consider the database \( DB = \{ \text{fish, beef, red-wine, white-wine, pie, ice-cream} \} \) and the following preference rules \( \Phi^* \):

\[
\begin{align*}
\rho_1 &: \text{pie} \gg \text{ice-cream} \leftarrow \\
\rho_2 &: \text{red-wine} \gg \text{white-wine} \leftarrow \text{fish} \\
\rho_3 &: \text{white-wine} \gg \text{red-wine} \leftarrow \text{beef}
\end{align*}
\]

The set \( \Phi^*_g \) consists, without considering useless rules, of a unique layer \( \Phi^*_g[1] = \{ \rho_1, \rho_2, \rho_3 \} \). The application of \( \Phi^*_g[1] \) to \( DB \) gives the set \( \Phi^*_g[1](DB) = \{ \text{fish, beef, red-wine, white-wine} \} \).

**Definition 8** Let \( DB \) be a database and let \( Q = \langle q, \mathcal{P}, \Phi \rangle \) be a prioritized query and \( \langle \mathcal{P}_1, \ldots, \mathcal{P}_k \rangle \) be the standard stratification of \( \mathcal{P} \). The application of \( \mathcal{P} \) and \( \Phi \) to \( DB \) is defined as follows: \( M_0 = \Phi^*_g[\mathcal{P}_0](DB) \) and for each \( i \) in \([1..k]\), \( M_i = \Phi^*_g[\mathcal{P}_i](M_{i-1}) \).

The answer to the query \( Q \) over the database \( DB \), denoted as \( Q(DB) \), is given by \( M_k[q] \).

Our proposal is sound, i.e. for each ground preference rule \( A \gg C \in \Phi^*_g \), if \( M_k \models (\text{body} \land A) \) then \( M_k \not

**4 CONCLUSIONS**

This paper has introduced prioritized queries, a form of queries well-suited for expressing preferences among tuples either belonging to the source database or derived by means of the program specified in the query. It has been shown that prioritized queries are well-suited to express queries wherein we are interested only in preferred tuples. A stratified semantics for computing prioritized queries has been presented where the program \( \mathcal{P} \) is partitioned into strata and the preference rules associated to each stratum of \( \mathcal{P} \) are divided into layers; a query is evaluated by computing one stratum at time and by applying the preference rules, one layer at time. The computational complexity of computing prioritized queries remains polynomial.

**REFERENCES**


