Algebraic Optimization of Relational Preference Queries

Werner Kießling, Bernd Hafenrichter

Report 2003-1 Februar 2003

Algebraic Optimization of Relational Preference Queries

Werner Kießling, Bernd Hafenrichter

Institute of Computer Science, University of Augsburg
Universitätsstraße 14, D-86159 Augsburg, Germany
{kiessling, hafenrichter}@informatik.uni-augsburg.de

Abstract

The design and implementation of advanced personalized database applications requires a preference-driven approach. Representing preferences as strict partial orders is a good choice in most practical cases. Therefore the efficient integration of preference querying into standard database technology is an important issue. We present a novel approach to relational preference query optimization based on algebraic transformations. A variety of new laws for preference relational algebra is presented. This forms the foundation for a preference query optimizer applying heuristics like 'push preference'. A prototypical implementation and a series of benchmarks show that significant performance gains can be achieved. In summary, our results give strong evidence that by extending relational databases by strict partial order preferences one can get both: good modeling capabilities for personalization and good query runtimes. Our approach extends to recursive databases as well.

1. Introduction

Preferences are an integral part of our private and business–related lives. Thus preferences must be a key element in designing personalized applications and Internet-based information systems. Personal preferences are often expressed in the sense of wishes: Wishes are free, but there is no guarantee that they can be satisfied at all times. In case of failure for the perfect match people are often prepared to accept worse alternatives or to negotiate compromises. Thus preferences in the real world require a paradigm shift from exact matches towards *match-making*, which means to find the best possible matches between one's wishes and the reality. In other words, preferences are *soft constraints*. On the other hand, declarative query languages like SQL don't offer convenient ways to express preferences. This deficiency has been the source for inadequate database support in many important application areas, in particular for search engines for e-commerce or m-commerce. As pointed out in [13, 17] extending SQL or XML by preferences will enable better personalized search engines that don't suffer from the notorious empty-result and flooding effects.

Preferences have played a big role in other academic disciplines for many decades, notably within the economic and social sciences, in particular for multi-attribute decision-making in operations research ([9, 12]). The first encounter of preferences in databases in 1987 is due to [20]. Despite the undeniable importance of preferences for real world applications preference research in databases did not receive a more wide-spread attention until around 2000 ([1, 4, 5, 11, 22]). Starting already in 1993 there has been the long-term research vision and endeavor of "*It's a Preference World*" at the University of Augsburg. Salient milestones so far include the design and implementation of *Datalog-S* ([14, 18]), extending recursive deductive databases by preference queries. By 1997 the experiences gained from Datalog-S inspired the design of *Preference SQL,* its first commercial product release being in 1999 ([17]). These experiences have been compiled into a comprehensive framework for preferences in database systems in [13]. Preferences are modeled as *strict partial orders*, providing also a unifying framework for approaches of other research groups like those cited above. As an essential feature the use of *intuitive preference constructors* is promoted.

In this paper we focus on the critical issue of *preference query performance*. In particular, we investigate the challenge of *algebraically* optimizing preference queries in relational databases. To set the stage, in section 2 we revisit the preference framework from [13]. In section 3 we introduce *preference relational algebra* and discuss architectural aspects for a preference query optimizer. Novel results for algebraic optimization of preference queries are presented in section 4, followed by performance experiments in section 5. Related works in section 6 and a summary and outlook in section 7 ends this paper.

2 The Preference Query Model

2.1 Strict Partial Order Preferences

People express their wishes intuitively in the form "*I like A better than B*". Mathematically such preferences are *strict partial orders*. Let us revisit those concepts of this preference model from [13] that are important here.

Let $A = \{A_1, A_2, \ldots, A_k\}$ denote a set of attributes A_j with domains $\mathrm{dom}(A_j)$. Considering the order of components within a Cartesian product as irrelevant, we define:

- $\mathrm{dom}(A) = \times_{A_j \in A} \mathrm{dom}(A_j)$
- A ***preference*** *P* is a strict partial order $\mathbf{P = (A, <_P)}$, where $<_P \subseteq \mathrm{dom}(A) \times \mathrm{dom}(A)$.
- "$x <_P y$" is interpreted as *"*I like y better than x*"*.

For ease of use a choice of ***base preference constructors*** is assumed to be pre-defined. This choice is *extensible*, if required by the application domain. Commonly useful constructors include the following:

- For *categorical* attributes:
  `POS, NEG, POS/POS,POS/NEG, EXP`
- For *numerical* attributes:
  `AROUND, BETWEEN, LOWEST, HIGHEST, SCORE`

`POS` specifies that a given set of values *should be* preferred. Conversely, `NEG` states a set of disliked values *should be* avoided if possible. `POS/POS` and `POS/NEG` express certain combinations, `EXP` explicitly enumerates 'better-than' relationships.

`AROUND` prefers values closest to a stated value, `BETWEEN` prefers values closest to a stated interval. `LOWEST` and `HIGHEST` prefer lower and higher values, resp. `SCORE` maps attribute values to numerical scores, preferring higher scores.

Compound preferences can be gained inductively by ***complex preference constructors***:

- *Pareto* preferences: $\mathbf{P := P_1 \otimes P_2 \otimes \ldots \otimes P_n}$
  P is a combination of *equally important* preferences, implementing the Pareto-optimality principle.
- *Prioritized* preferences: $\mathbf{P := P_1 \;\&\; P_2 \;\&\; \ldots \;\&\; P_n}$
  P evaluates *more important* preferences earlier, similar to a lexicographical ordering. P_1 is most important, P_2 next, etc.
- *Numerical* preferences: $\mathbf{P := rank_F(P_1, P_2, \ldots, P_n)}$
  P combines `SCORE` preferences P_i by means of a numerical ranking function F.

Concerning the formal definitions of preference constructors the reader is referred to [13].

Example 1: *Preference constructors*

"Julia wishes to buy a used car. It *must* be a BMW. Moreover, it *should* have a red or black color and *equally important* is a price around 10,000. Highest fuel economy matters too, but is *less important*."

Wanting a BMW is a hard condition. Julia's preferences, i.e. soft conditions, can be extracted right away from this natural language description as follows:

```
P_Julia = (POS(color,{'red','black'})⊗
           AROUND(price,10000))
          & HIGHEST(fuel_economy)          ☼
```

Preference construction is inductively closed under strict partial order semantics ([13]). Due to a well-known theorem in [9] (see also [5]) numerical preferences have a *limited* expressiveness. Many strict partial order preferences cannot be described by numerical preference constructors only. Therefore the support of the full preference constructor spectrum as described is a practical necessity.

2.2 The BMO Query Model

Extending declarative query languages by preferences leads to *soft selection conditions*. To combat the empty-result and the flooding effects the ***Best-Matches-Only* (BMO)** query model has been proposed in [13]. Assuming a preference $P = (A, <_P)$ and a database relation R, BMO query answering conceptually works as follows:

- Try to find *perfect matches* in R wrt. P.
- If none exist, deliver *best-matching alternatives*, but *nothing worse.*

Efficient BMO query evaluation requires two new relational operators. Assuming a relation R where $A \subseteq$ attributes(R), we define:

- ***Preference selection*** $\boldsymbol{\sigma}$**[P](R):**
  $\sigma[P](R) := \{w \in R \mid \neg\exists v \in R : w[A] <_P v[A]\}$

A preference can also be evaluated in grouped mode, given some $B \subseteq$ attributes(R). According to [13] this can be expressed as a preference itself:

$$w <_{P \text{ groupby } B} v \quad \text{iff} \quad w[A] <_P v[A] \wedge w[B] = v[B]$$

- ***Grouped preference selection*** $\boldsymbol{\sigma}$**[P groupby B](R):**
  $\sigma[P \text{ groupby } B](R) :=$
  $\{w \in R \mid \neg\exists\, v \in R : w[A] <_P v[A] \;\wedge\; w[B] = v[B]\}$

$\sigma[P](R)$ and $\sigma[P \text{ groupby } B](R)$ can perform the match-making process as required by BMO semantics.

2.3 Practical Preference Query Languages

Existing implementations are Preference SQL for SQL environments and Preference XPATH for XML ([16]). Subsequently we will use Preference SQL syntax for our case studies. Preference queries are specified as follows:

```
SELECT      <projection list L>
FROM        <R1, …, Rn>
WHERE       <hard conditions H>
PREFERRING  <soft conditions P>
GROUPING    <B1, …, Bm>;
```

`SELECT-FROM-WHERE` is standard SQL, whereas `PREFERRING-GROUPING` cares for preferences.

Example 2: *A preference query and its BMO result*

“Under no circumstances Michael can afford to spend more than 35,000 Euro for a car. Other than that he wishes that the car *should be* a BMW or a Porsche, it *should be* around 3 years old and the color *shouldn’t be* red. All these preferences are *equally important* to him.“

Michael’s overall preference is specified by:

```
PMichael = POS(brand,{‘BMW’,‘Porsche’}) ⊗
           AROUND(age,3) ⊗
           NEG(color,{‘red’})
```

Given that the car database is organized as shown below, the Preference SQL query asking for Michael’s best matching cars is as follows:

```
SELECT u.price, c.brand, u.age, u.color
FROM   used_cars u, category c
WHERE  u.ref = c.ref AND
       u.price <= 35000
PREFERRING c.brand IN (‘BMW’, ‘Porsche’)
       AND u.age AROUND 3
       AND u.color NOT IN (‘red’);
```

Note that ‘AND’ in the WHERE-clause means Boolean conjunction, whereas ‘AND’ in the PREFERRING-clause denotes Pareto preference construction. Now let’s evaluate above query against this car database:

used_cars

ref	price	age	color	*sid*	car#
1	25,000	4	red	1	1
2	20,000	5	blue	2	2
3	30,000	3	yellow	3	3
4	27,000	4	blue	4	4
1	30,000	4	blue	1	5
4	15,000	6	red	5	6
5	40,000	3	black	6	7

category

ref	brand	type	horsepower
1	BMW	cabriolet	200
2	Renault	coupé	75
3	Fiat	coupé	45
4	Audi	avant	150
5	Porsche	cabriolet	350

Unfortunately the black Porsche is absolutely unaffordable and must be dismissed beforehand. From the two BMWs, both being top on brand and equal on age, the blue one dominates the red because of the better color. The Renault is dominated by the blue BMW, too. However, the Fiat and the blue BMW are unordered, since the BMW wins on brand, but looses on age. Lastly, both Audis are beaten by the BMWs. Thus the overall preference query result under BMO semantics is this:

price	brand	age	color
30,000	BMW	4	blue
30,000	Fiat	3	yellow

☼

3 Relational Preference Query Optimization

3.1 Preference Relational Algebra

For the scope of this paper we restrict our attention to the non-recursive relational case, although our preference model is applicable to the general case of recursive deductive databases as well ([14]).

Let ***preference relational algebra*** denote the following two sets of operations:

- Standard positive relational algebra:
  hard selection $\boldsymbol{\sigma_H(R)}$ given a Boolean condition H, projection $\boldsymbol{\pi(R)}$, union $\boldsymbol{R \cup S}$, Cartesian product $\boldsymbol{R \times S}$
- *Preference operations:*
  - preference selection $\boldsymbol{\sigma[P](R)}$
  - grouped preference selection $\boldsymbol{\sigma[P \text{ groupby } B](R)}$

Due to a theorem in [6] the *expressive power* of preference relational algebra is the same as classical relational algebra (i.e. positive relational algebra plus “\”). Consequently preference queries under BMO can be rewritten into relational algebra (which is done by Preference SQL, see [17]). It also implies that the optimization problem for preference relational algebra is not harder than for classical relational algebra, i.e. in principle preferences can be integrated into SQL with sufficient performance.

3.2 Operational Semantics of a Preference Query

Defining the semantics of a declarative query language in general requires a model-theoretic semantics (to capture the declarative aspects) and an equivalent fixpoint semantics (to capture the operational aspects of query evaluation). For Preference SQL this task can be embedded into the larger framework of Datalog-S, employing *subsumption models* and *subsumption fixpoints*, resp. ([18]).

Since our focus is here on algebraic optimization of *relational* preference queries, we can exploit the equivalence of relational algebra and preference relational algebra to define the operational semantics. Let’s consider a preference query Q in Preference SQL syntax. Then the ***operational semantics*** of Q is defined as:

If <grouping clause does not exist>
then $\boldsymbol{\pi_L (\sigma[P](\sigma_H (R_1 \times \ldots \times R_n)))}$
else $\boldsymbol{\pi_L (\sigma[P \text{ groupby}\{B_1, \ldots, B_m\}] (\sigma_H(R_1 \times \ldots \times R_n)))}$

This canonically extends the familiar relational case ([23]). Referring back to our example 2, the operational semantics of this preference query is as follows.

Example 2 (cont'd): *Operational semantics*

```
πu.price, c.brand, u.age, u.color
  (σ[POS(c.brand,{'BMW','Porsche'}) ⊗
     AROUND(u.age,3) ⊗ NEG(color,{'red'})]
    (σu.ref = c.ref ∧ u.price <= 35000
      (used_cars u × category c)))         ☼
```

Such an initial preference relational algebra expression will be subject to algebraic optimization methods developed subsequently.

3.3 Architectural Design Issues

Extending an existing SQL implementation by preference queries requires some crucial design decisions for the preference query optimizer.

➢ **The loosely coupled pre-processor architecture:**
Preference queries are processed as sketched in figure 1 by rewriting them to standard SQL and submitting them to the SQL database. The current version of Preference SQL follows such a loose coupling, achieving acceptable performance in many e-commerce applications ([17]).

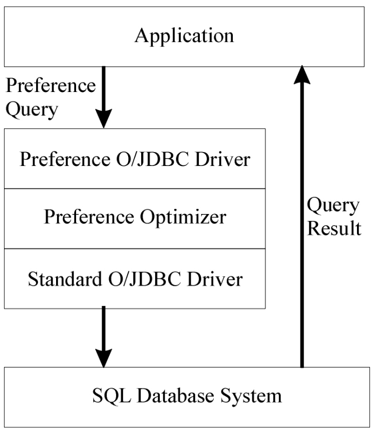


Figure 1: Pre-processor approach for preference queries

➢ **The tightly coupled architecture:**
Integrating the preference query optimizer and the SQL optimizer more tightly as sketched in figure 2 promises an even much better performance. A practical obstacle might be that this implementation requires the close cooperation with a specific SQL database manufacturer.

Here the optimization problem for preference queries can be mapped onto preference relational algebra. In fact we can follow the classical paradigm of database query optimization ([23]), given a preference query Q:

1. The operational semantics of Q defines an *initial operator tree* T_{start}, whose nodes are the operators from preference relational algebra (cmp. figure 3).
2. T_{start} is optimized using some set of *algebraic transformation laws*. Which laws to apply, and in what order, has to be controlled by some *heuristics* that intelligently prune the usually exponentially large search space. Let T_{end} denote the final tree.
3. The preference relational algebra operators in T_{end} must be mapped to efficient evaluation algorithms. If there are several choices, a *cost-based* optimization has to decide.

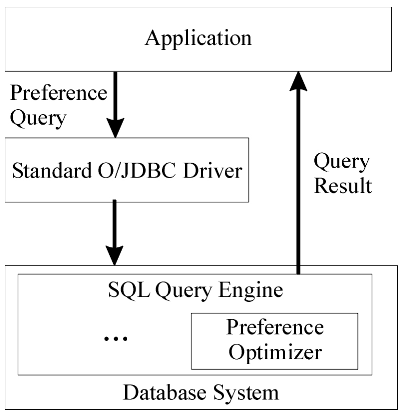


Figure 2: Tightly coupled preference query evaluation

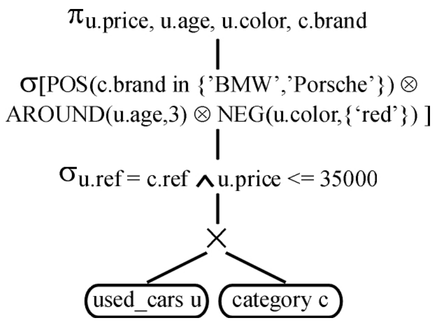

πu.price, u.age, u.color, c.brand
σ[POS(c.brand in {'BMW','Porsche'}) ⊗
AROUND(u.age,3) ⊗ NEG(u.color,{'red'})]
σu.ref = c.ref ∧ u.price <= 35000
×
used_cars u
category c

Figure 3: Initial operator tree for example 2 (cont'd)

4 Preference Relational Algebra Laws

Among the successful heuristic strategies of algebraic relational query optimization are '*push hard selection*' and '*push projection*'. We extend this idea by developing transformation laws for preference relational algebra that allow us to perform '*push preference*' within operator trees.

4.1 Transformation Laws

Some annotations of the subsequent theorems refer to left-to-right 'push' transformations. We use attr(R) to denote all attributes of a relation R. The proofs are given in the appendix.

Theorem L1: *Push preference over projection*

Let $P = (A, <_P)$ and $A, X \subseteq attr(R)$.

a) $\sigma[P](\pi_X(R)) = \pi_X(\sigma[P](R))$ if $A \subseteq X$
b) $\pi_X(\sigma[P](\pi_{X \cup A}(R))) = \pi_X(\sigma[P](R))$ otherwise

Theorem L2: *Push preference over Cartesian product*

Let $P = (A, <_P)$ and $A \subseteq attr(R)$.
$\sigma[P](R \times S) = \sigma[P](R) \times S$

Theorem L3: *Push preference over union*

Let $P = (A, <_P)$ and $A \subseteq attr(R) = attr(S)$.
$\sigma[P](R \cup S) = \sigma[P](\sigma[P](R) \cup \sigma[P](S))$

Theorem L4 : *Split prioritization into grouping*

Let $P_1 = (A_1, <_{P1})$, $P_2 = (A_2, <_{P2})$ and $A_1, A_2 \subseteq attr(R)$.
$\sigma[P_1 \& P_2](R) = \sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R))$

Corollary 1: *Simplify prioritization*

$\sigma[P_1 \& P_2](R) = \sigma[P_2](\sigma[P_1](R))$ if P_1 is a chain

`LOWEST` and `HIGHEST` preferences are chains, i.e. total orders. If P_1 and P_2 are chains, $P_1 \& P_2$ is a chain ([13]).

Recalling the definition of the operational semantics of a preference query Q the reader might have asked herself or himself, why σ_H is applied *before* $\sigma[P]$ and not vice versa, and whether this would make a difference. It turns out that $\sigma[P]$ can be interchanged with σ_H only under a rather strong precondition. (This is also one reason why Preference SQL in addition supports the `'BUT ONLY'` clause for hard selections *after* preference selection [17].)

Theorem L5: *Push preference over hard selection*

$\sigma[P](\sigma_H(R)) = \sigma_H(\sigma[P](R))$ iff
$\forall w \in R: (H(w) \wedge \exists v \in R: w[A] <_P v[A]$ implies $H(v))$

Corollary 2: *Special cases of L5*

Let $P_1 := LOWEST(A)$, $P_2 := HIGHEST(A)$.
a) $\sigma[P_1](\sigma_{A <= c}(R)) = \sigma_{A <= c}(\sigma[P_1](R))$
b) $\sigma[P_2](\sigma_{A >= c}(R)) = \sigma_{A >= c}(\sigma[P_2](R))$

Since a relational join is a hard selection over a Cartesian product, pushing $\sigma[P]$ over a join is more difficult.

Theorem L6 *Push preference over a join*

Let $P = (A, <_P)$, $A \subseteq attr(R)$ and $X \subseteq attr(R) \cap attr(S)$.
a) $\sigma[P](R \bowtie_{R.X = S.X} S) = \sigma[P](R) \bowtie_{R.X = S.X} S$,
if each tuple in R has at least one join partner in S

Let $R \ltimes_{R.X = S.X} S$ denote a semi-join operation.
b) $\sigma[P](R \bowtie_{R.X = S.X} S) =$
$\sigma[P](R \ltimes_{R.X = S.X} S) \bowtie_{R.X = S.X} S$
c) $\sigma[P](R \bowtie_{R.X = S.X} S)) =$
$\sigma[P](\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S)$

Further let $B \subseteq attr(S)$.
d) $\sigma[P \text{ groupby } B](R \bowtie_{R.X = S.X} S) =$
$\sigma[P \text{ groupby } B](\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S))$

Note that as a special case law L6a is applicable, if R.X is a *foreign key* referring to S.X. If no such meta-information is available, then L6b explicitly computes all join partners. Sometimes a preference must be *split* before a part of it can be pushed over a join.

Theorem L7: *Split Pareto preference and push over join*

Let $P_1 = (A_1, <_{P1})$ where $A_1 \subseteq attr(R)$, $P_2 = (A_2, <_{P2})$ where $A_2 \subseteq attr(S)$. Further let $X \subseteq attr(R) \cap attr(S)$.
$\sigma[P_1 \otimes P_2](R \bowtie_{R.X = S.X} S) =$
$\sigma[P_1 \otimes P_2] (\sigma[P_1 \text{ groupby } X](R) \bowtie_{R.X = S.X} S)$

Theorem L8: *Split prioritization and push over join*

Let $P_1 = (A_1, <_{P1})$ where $A_1 \subseteq attr(R)$, $P_2 = (A_2, <_{P2})$ where $A_2 \subseteq attr(R) \cup attr(S)$ and let $X \subseteq attr(R) \cap attr(S)$.
a) $\sigma[P_1 \& P_2](R \bowtie_{R.X = S.X} S) =$
$\sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R) \bowtie_{R.X = S.X} S)$,
if each tuple in R has at least one join partner in S
b) $\sigma[P_1 \& P_2](R \bowtie_{R.X = S.X} S) =$
$\sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R \ltimes_{R.X = S.X} S) \bowtie_{R.X = S.X} S)$

Concerning the precondition on join partners the same remark as for L6 applies here too.

Next we state some handy laws to simplify complex grouped preference selections.

Theorem L9: *Simplify grouped preference selections*

Let $P = (A, <_P)$, $P_1 = (A_1, <_{P1})$, $P_2 = (A_2, <_{P2})$ and $A, B, B_1, B_2 \subseteq attr(R)$.
a) $\sigma[(P \text{ groupby } B_1) \text{ groupby } B_2](R) =$
$\sigma[P \text{ groupby } (B_1 \cup B_2)](R)$
b) $\sigma[(P_1 \text{ groupby } B) \& P_2](R) =$
$\sigma[(P_1 \& P_2) \text{ groupby } B](R)$
c) $\sigma[P_1 \otimes (P_2 \text{ groupby } B)](R) =$
$\sigma[(P_1 \otimes P_2) \text{ groupby } B](R)$
d) $\sigma[(P_1 \text{ groupby } B_1) \otimes (P_2 \text{ groupby } B_2)](R) =$
$\sigma[(P_1 \otimes P_2) \text{ groupby } (B_1 \cup B_2)](R)$
e) $\sigma[P \text{ groupby } B](R) = R$ if B is key in R

The strong precondition in theorem L5 is frequently violated. However, in some cases we can state other useful laws for cascades of preference and hard selections.

Theorem L10: *Cascade of preference and hard selection*

a) Let $P = (A, <_P)$ and $A, B \subseteq attr(R)$:
$\sigma[P](\sigma_{B = c}(R)) = \sigma_{B = c}(\sigma[P \text{ groupby } B](R))$
$\sigma[P](\sigma_{A = c}(R)) = \sigma_{A = c}(R)$
b) Let $P := BETWEEN(A, [c_1, c_2])$:
$\sigma[P](\sigma_{A >= c}(R)) = \sigma[LOWEST(A)](\sigma_{A >= c}(R))$, if $c > c_2$
$\sigma[P](\sigma_{A <= c}(R)) = \sigma[HIGHEST(A)](\sigma_{A <= c}(R))$, if $c < c_1$
$\sigma[P](\sigma_{A >= c1 \wedge A <= c2}(R)) = \sigma_{A >= c1 \wedge A <= c2}(R)$

Let H := '$A = c_1 \vee ... \vee A = c_n$', H-set := $\{c_1, ..., c_n\}$.
c) Let P := POS(A, Pos-set), Pos-set = $\{v_1, ..., v_m\}$:
if H-set $\cap$ Pos-set $= \emptyset$ or H-set $\subseteq$ Pos-set
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$
d) Let P := NEG(A, Neg-set), Neg-set = $\{v_1, ..., v_m\}$:
if H-set $\cap$ Neg-set $= \emptyset$ or H-set $\subseteq$ Neg-set

then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

e) Let P = POS/NEG(A, POS-set; NEG-set), POS-set=$\{v_1, ..., v_m\}$, NEG-set = $\{v_{m+1}, ..., v_{m+n}\}$:
If H-set $\subseteq$ POS-set or H-set $\subseteq$ NEG-set or H-set $\cap$ (POS-Set $\cup$ NEG-Set) = Ø
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

f) Let P = POS/POS(A, POS1-set; POS2-set), POS1-set=$\{v_1, ..., v_m\}$, POS2-set = $\{v_{m+1}, ..., v_{m+n}\}$:
If H-set $\subseteq$ POS1-set or H-set $\subseteq$ POS2-set or H-set $\cap$ (POS1-Set $\cup$ POS2-Set) = Ø
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

Since AROUND is a preference *sub-constructor* of BETWEEN ([13]) we get another corollary for free.

Corollary 3: AROUND *inherits from* BETWEEN

Theorem L10b holds also for P := AROUND(A, z).

Theorem L11: *Commutativity and associativity*

a) $\sigma[P_1 \otimes P_2](R) = \sigma[P_2 \otimes P_1](R)$
b) $\sigma[(P_1 \otimes P_2) \otimes P_3](R) = \sigma[P_1 \otimes (P_2 \otimes P_3)](R)$
c) $\sigma[(P_1 \,\&\, P_2) \,\&\, P_3](R) = \sigma[P_1 \,\&\, (P_2 \,\&\, P_3)](R)$

L11 follows directly from [13]. It also implies that laws on Pareto preferences like L7 are commutative.

Note that since 'P groupby B' is a preference itself, in each of the previous laws a preference may be grouped or not, unless explicitly required like e.g. in L9. For instance, given a grouping preference L1a reads as follows:

$\pi_X(\sigma[P \text{ groupby } B](R)) =$
$\sigma[P \text{ groupby } B](\pi_X(R))$ if A, B $\subseteq$ X

4.2 Integration with a Relational Query Optimizer

Because preference relational algebra extends relational algebra, we can construct a ***preference query optimizer*** as an extension of a classical relational query optimizer. Importantly, we can inherit all familiar laws from relational algebra ([23]). Thus we can apply well-established heuristics aiming to reduce the sizes of intermediate relations, e.g. '*push hard selection*' and '*push projection*'. Let's consider this basic hill-climbing algorithm ([23]), given a suitable repertoire of relational algebra transformations:

```
Algorithm Pass-1(T):
{ update T:
    Step 1-1: <split hard selections>;
    Step 1-2: <push hard selections as far as possible>;
    Step 1-3: <push projections as far as possible>;
    Step 1-4: <combine hard selections and Cartesian
              products into joins>; return T}
```

Expanding the given repertoire of relational transformations by our new laws L1 to L11 raises the issue of how to integrate them into above procedure. In particular, we want to add the heuristic strategy of '*push preference*'. Mixing the new transformations into the given optimization strategy would give the highest flexibility. But from a software engineering point of view this might be problematic. A possibly less flexible, but much less invasive way is to extend the existing relational query optimizer by a second pass as illustrated below.

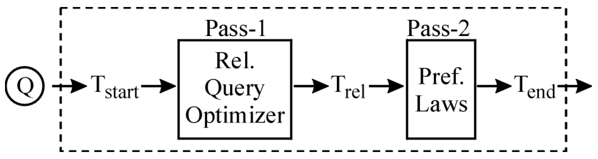


Figure 4: Two-pass principle for algebraic optimization

Given a preference query Q, its initial operator tree T_{start} is submitted to Pass-1, beginning its work at the subtree below the preference node. The output tree T_{rel} is handed over to our novel Pass-2, working as follows:

```
Algorithm Pass-2(T):
{ repeat <traverse T in a depth-first way>
  { Step 2-1: // simplify prioritization
        apply corollary 1;
    Step 2-2: // push preference over projection,
                 Cartesian product or union
        apply L1a; L1b; L2; L3;
    Step 2-3: // push preference over join
        apply L6a; L6b;
    Step 2-4: // split preference and push over join
        apply L7; L9e; L8a; L8b };
  Step 2-5: // push projections as far as possible
        apply Step 1-3 including L1a, L1b in 'pull' mode;
  return T}
```

Pass-2(T) checks, if 'push preference' transformations are applicable and updates T accordingly. Finding a 'good' ordering of transformations is as usual a difficult, heuristic task. Starting with corollary 1 in Step 2-1should always be ok. Also in any case Step 2-3 should come before Step-4 because of a better filter effect. L11a-c are implicitly utilized by Step 2-4 to localize suitable split points. L9e undoes apparently harmful firings of L7. Note that L7 and L8a,b relate to different situations, hence their relative order within Step 2-4 does not matter. T is repeatedly traversed until no more preferences can be pushed. To prevent an infinite firing of laws like L7, proper context information is maintained. Finally in Step 2-5 we re-invoke Step 1-3, since Pass-1 started its job only below the preference node in T_{start}. Then Pass-2(T) terminates and returns the final operator tree T_{end}.

Note that laws L9a-d, L10a-d are not implemented for the time being. Laws L4, L5, L6c, d are required for proof purposes. Now we demonstrate the potential of such a preference query optimizer by two practical case studies.

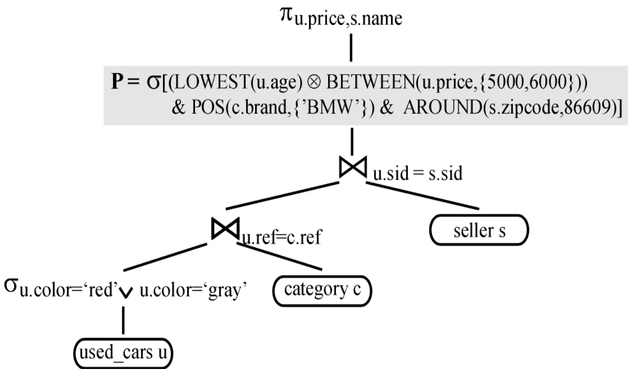
4.3 Case Studies

Let's revisit our example 2, adding another relation:

```
  seller(sid,name,street,zipcode,city)
```
In `used_cars` we assume that *ref* and *sid* are foreign keys from `category` and `seller`, resp., and Car# is primary key.

Example 3: *Complex preference query Q_1*

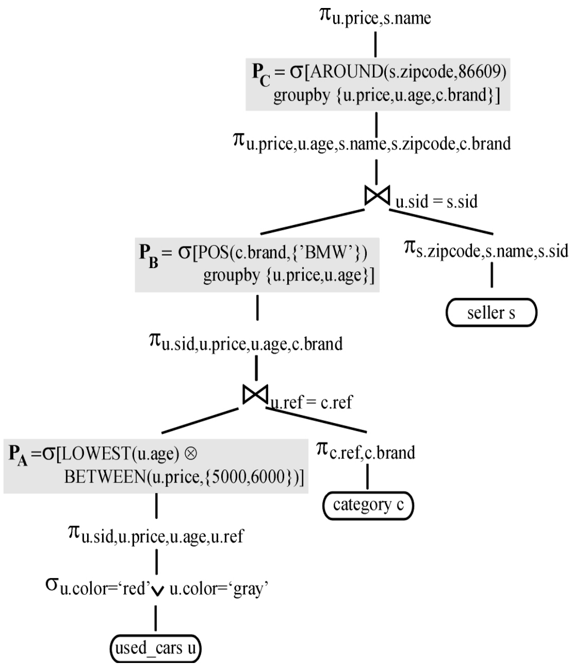
```
SELECT s.name, u.price
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref AND
      (u.color = 'red' OR u.color = 'gray')
PREFERRING
      (LOWEST u.age AND
       u.price BETWEEN 5000, 6000)
      PRIOR TO c.brand IN ('BMW')
      PRIOR TO s.zipcode AROUND 86609;
```



πu.price,s.name
P = σ[(LOWEST(u.age) ⊗ BETWEEN(u.price,{5000,6000})) & POS(c.brand,{'BMW'}) & AROUND(s.zipcode,86609)]
⋈u.sid = s.sid
⋈u.ref=c.ref
seller s
σu.color='red'∨ u.color='gray'
category c
used_cars u

Figure 5: T_{rel} after pass 1 (complex query)

The tree T_{rel} as output by Pass-1 is depicted in figure 5. Given T_{rel} Pass-2 performs the following sequence of preference transformations to produce the final operator tree T_{end} in figure 6: **`L8a; L8a; L1b; L1a; L1a`**



πu.price,s.name
PC = σ[AROUND(s.zipcode,86609) groupby {u.price,u.age,c.brand}]
πu.price,u.age,s.name,s.zipcode,c.brand
⋈u.sid = s.sid
PB = σ[POS(c.brand,{'BMW'}) groupby {u.price,u.age}]
πs.zipcode,s.name,s.sid
seller s
πu.sid,u.price,u.age,c.brand
⋈u.ref = c.ref
PA =σ[LOWEST(u.age) ⊗ BETWEEN(u.price,{5000,6000})]
πc.ref,c.brand
category c
πu.sid,u.price,u.age,u.ref
σu.color='red'∨ u.color='gray'
used_cars u

Figure 6: T_{end} after pass 2 (complex query)

Let's compare T_{end} vs. T_{rel}:
- *Join costs*: The lower join's left operand in T_{end} is reduced by the preference selection marked P_A in figure 6. This in turn, intensified by the preference selection P_B, reduces the size of the upper join's left operand in T_{end}.
- *Preference costs*: P_A is simpler than the original P in T_{rel}, but it's still a compound preference. However, both P_A and P_C are simpler, i.e. grouped *base* preferences.

Now the tradeoff is more apparent. Our heuristics of 'push preference' -in particular over joins- will pay off, if the savings for join computation outweigh the preference costs of P_A, P_B, P_C vs. the original P. ☼

S*kyline queries* as studied in [4] are a sub-class of Pareto preferences. Here is one in Preference SQL syntax.

Example 4: *Skyline query Q_2*

```
 SELECT s.name, u.price
 FROM   used_cars u, category c, seller s
 WHERE  u.sid = s.sid AND u.ref = c.ref AND
        c.brand = 'BMW'
 PREFERRING LOWEST u.age AND
   LOWEST u.price AND HIGHEST c.horsepower;
```

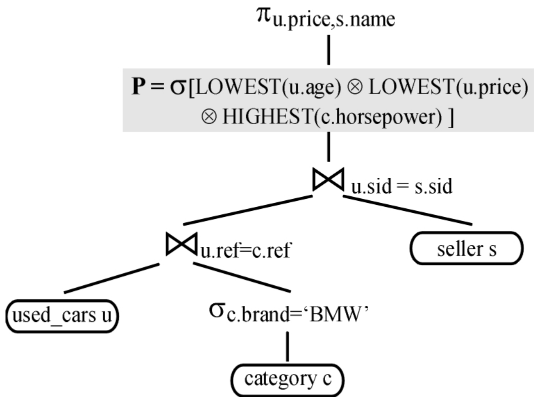
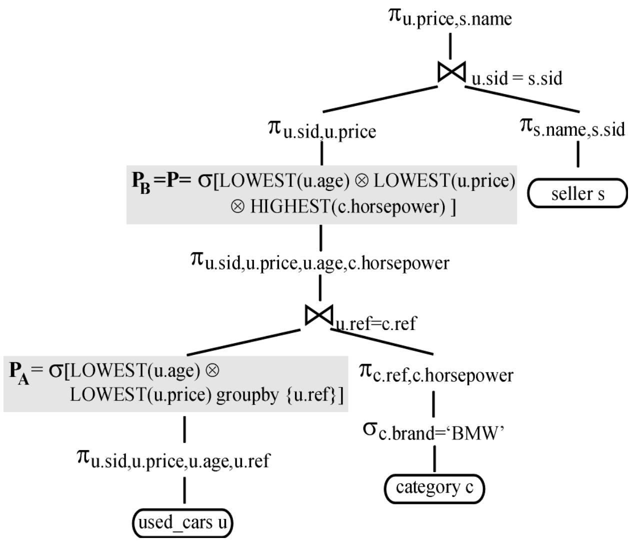


Figure 7: T_{rel} after pass 1 (skyline query)

The tree T_{rel} output by Pass-1 is depicted in figure 7. Given T_{rel} Pass-2 performs these preference transformations to generate the final tree T_{end} depicted in figure 8:
`L6a; L7; L7; L9e; L1b; L1a`

Here a comparisons of T_{end} vs. T_{rel} yields:
- *Join costs*: The same arguments as above apply.

Preference costs: The original skyline P has been pushed and split into two skylines P_A and P_B = P. ☼



$\pi_{u.price,s.name}$
$\bowtie_{u.sid = s.sid}$
$\pi_{u.sid,u.price}$
$\pi_{s.name,s.sid}$
P_B =P= σ[LOWEST(u.age) ⊗ LOWEST(u.price) ⊗ HIGHEST(c.horsepower)]
seller s
$\pi_{u.sid,u.price,u.age,c.horsepower}$
$\bowtie_{u.ref=c.ref}$
P_A = σ[LOWEST(u.age) ⊗ LOWEST(u.price) groupby {u.ref}]
$\pi_{c.ref,c.horsepower}$
$\sigma_{c.brand='BMW'}$
$\pi_{u.sid,u.price,u.age,u.ref}$
category c
used_cars u

Figure 8: T_{end} after pass 2 (skyline query)

5 Performance Evaluation

Finally we present performance experiments from a prototype implementation of our preference query optimizer, employing the two-pass engineering principle for algebraic transformations. Since we did not have quick access to a commercial SQL query engine to tightly integrate our novel optimization techniques we decided to simulate it. For the sake of *rapid prototyping* we built a Java middleware on top of Oracle 9i. This task has been facilitated by the use of the XXL Java-library ([3]), offering a collection of relational algebra operators.

5.1. Prototype Implementation

Figure 9 illustrates how a preference query Q is evaluated:

- Q is mapped onto its initial operator tree T_{start}.
- T_{start} is algebraically optimized, employing Pass-1 with output T_{rel} and Pass-2 with final output T_{end}.
- T_{end} is submitted to an evaluation middleware, utilizing XXL for relational operations and providing own Java implementations of the preference operators σ[P](R) and σ[P groupby B](R).
- All persistent database relations are managed by Oracle 9i, accessed from XXL via JDBC.

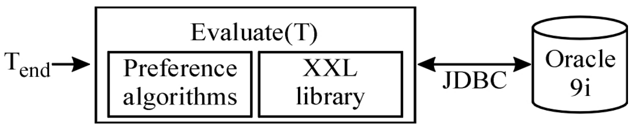


Figure 9: Rapid prototyping for performance studies

The development of efficient evaluation algorithms for σ[P](R) is considerably facilitated by the constructor-based approach of our preference model. Depending on the preference constructor P we can efficiently *custom-design* the evaluation method for σ[P](R) as follows.

Most specific evaluation algorithms for σ[P](R):
We generalized the basic *block nested loop* algorithm BNL, investigated by [4] in the context of skyline queries, to arbitrary strict partial order preferences. However, the subsumption test '$x <_P y$' within BNL can be simplified significantly, if P is known to have special properties. This is the case for each of our *base* preference constructors (cmp. section 2.1). Therefore we have implemented the following evaluation policy for σ[P](R):

- If P is a *complex* preference (i.e. constructed e.g. by '⊗' and-or '&'), then the general BNL applies.
- Otherwise, if P is a base preference constructor, then a specialized algorithm for this P has to be chosen.

BNL is known to degrade to $O(n^2)$ in the worst case. In contrast, the complexity of specialized algorithms for base preference constructors is much better: *O(n)* if no index support is provided, otherwise *O(log n)*.

Considering σ[P groupby B](R), directly exploiting its formal definition as a complex preference and applying BNL would risk an $O(n^2)$ performance penalty. More efficient is to implement the grouping effect e.g. by sorting and to invoke the most specific algorithm for P on each group. Therefore if P is a base preference, an *O(n log n)* behavior can be accomplished for σ[P groupby B](R).

The procedure `Evaluate(T)` in figure 9 traverses T and maps subtrees to XXL methods or *most specific* preference algorithms, returning JDBC ResultSets. Pipelining of XXL and of preference algorithms is enforced as much as possible. Expressions of type $\pi(\sigma_H(R))$, possibly extended by semi-joins generated by laws L6b or L8b, are evaluated by Oracle if R is a database relation. To expedite the evaluation of grouped preferences on $\pi(\sigma_H(R))$ the ResultSet is returned ordered by B. Moreover, we replaced the nested-loops join of XXL by a more efficient hash join. Doing so, our heuristics of 'push preference' has a tougher task to prove its benefits.

5.2 Performance Results

We have built up a test suite for performance evaluation. Given a preference query Q, we carried out the following performance measurements in our rapid prototype:

- Runtime $\mathbf{t_{end}}$ for evaluating T_{end} (in sec.)
- Runtime $\mathbf{t_{rel}}$ for evaluating T_{rel} (in sec.)

As an indicator for the optimization impact of our new approach we choose the *speedup factor* **SF1:=** $\mathbf{t_{rel} / t_{end}}$.

The experiments were performed on a standard PC (2 GHz CPU, 512 MB main memory) running XP and JDK 1.3. The relation `seller` has 5000 tuples, `category` has 1000, while `used_cars` varies from 1000 to 50000.

All data have been generated synthetically. Values are uniformly distributed. Attributes are uncorrelated except that higher `age` anti-correlates with higher `price`.

We present selected characteristic tests, beginning with our running examples 3 and 4. Please note that the operator trees for the examples are given in appendix 2.

Example 3 (cont'd): *Performance results for Q_1*

For the transformation sequence and the pushed and split preferences P_A, P_B and P_C please refer back to example 3.

used_cars	1000	5000	10000	50000
BMO size	15	59	138	518
$\mathbf{t_{rel}}$	0.7	0.8	1.1	4.3
$\mathbf{t_{end}}$	0.2	0.3	0.5	2.8
SF1	**3.5**	**2.7**	**2.2**	**1.5**

The original P was split and pushed twice by L8a. P_B and P_C are grouped base preferences and can be evaluated reasonably fast. The net effect is a sizeable performance gain as indicated by SF1. ☼

Example 4 (cont'd): *Performance results for Q_2*

For the transformation sequence and the pushed and split preferences P_A and P_B please refer back to example 4.

used_cars	1000	5000	10000	50000
BMO size	26	42	78	91
$\mathbf{t_{rel}}$	0.7	0.9	1.3	4.4
$\mathbf{t_{end}}$	0.2	0.5	0.8	3.3
SF1	**3.5**	**1.8**	**1.6**	**1.3**

The original skyline P was pushed by L6a and split producing the skylines P_A and P_B = P. Though both are evaluated by the general BNL algorithm, their early filter effect dominates as reflected by SF1. ☼

Example 5: *Performance results for Q_3*

Q_3 is derived from Q_2 by changing the preference into:

P = LOWEST(u.age) &
(LOWEST(u.price) ⊗ HIGHEST(c.horsepower))

The preference transformations carried out are:

```
Cor1; L6a; L6b; L6a; L7; L7; L9e;
L1b; L1a; L1b
```

Pushed and split preferences in T_{end} are:

P_A = LOWEST(u.age)
P_B = LOWEST(u.price) groupby {u.ref}
P_C = LOWEST(u.price) ⊗ HIGHEST(c.horsepower)

used_cars	1000	5000	10000	50000
BMO size	3	2	6	4
$\mathbf{t_{rel}}$	0.7	0.8	1.0	3.5
$\mathbf{t_{end}}$	0.2	0.2	0.3	0.4
SF1	**3.5**	**4.0**	**3.3**	**8.8**

The early filter effect accomplished by the highly selective and efficient base preference P_A is the key factor for these substantial SF1-gains. ☼

Example 6: *Performance results for Q_4*

Q_4 is derived from Q_2 by changing the preference into:
P = LOWEST(u.age) & LOWEST(u.price)

The preference transformations are:

```
Cor1; L6a; L6b; L6a; L6b; L1a; L1b
```

Pushed and split preferences in T_{end}:
P_A = LOWEST(u.age), P_B = LOWEST(u.price)

used_cars	1000	5000	10000	50000
BMO size	1	1	1	1
$\mathbf{t_{rel}}$	0.6	0.8	1.0	3.9
$\mathbf{t_{end}}$	0.2	0.2	0.2	0.4
SF1	**3.0**	**4.0**	**5.0**	**9.8**

Query Q_4 is simpler than the previous Q_3. Here P can be split and pushed even into two highly selective and efficient base preferences P_A and P_B, resulting in a high performance speedup. ☼

Example 7: *Performance results for Q_5*

```
SELECT s.name, u.price as 'pr',
       u.age as 'ag', c.horsepower AS 'hp'
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref AND
      (u.color = 'red' OR u.color = 'gray')
UNION
SELECT s.name, u.price as 'pr',
       u.age as 'ag', c.horsepower AS 'hp'
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref AND
       c.brand = 'BMW'
PREFERRING (LOWEST ag AND LOWEST pr)
           PRIOR TO HIGHEST hp;
```

The sequence of preference transformations is:

```
L3; L1a; L6a; L8a; L1a; L6a; L8b;
L1a; L1a; L1a; L1a
```

Pushed and split preferences P_A, P_B, P_C and P_D in T_{end}:
P_A = LOWEST(ag) ⊗ LOWEST(pr) = P_C
P_B = HIGHEST(hp) groupby {pr, ag} = P_D

used_cars	1000	5000	10000	50000
BMO size	17	24	28	29
$\mathbf{t_{rel}}$	1.2	1.6	1.9	5.4
$\mathbf{t_{end}}$	0.4	0.5	0.6	1.3
SF1	**3.0**	**3.2**	**3.2**	**4.2**

L3 is the catalyst for all pushes and splits. P_B, P_D can be evaluated reasonably fast, hence good SF1-values. ☼

Example 7: *Performance results for Q_6*

```
SELECT u.price FROM used_cars u, seller s
WHERE  u.sid = s.sid
PREFERRING u.price BETWEEN 5000, 30000 AND
           u.age AROUND 25 AND
           s.zipcode BETWEEN 20000, 50000;
```

This query has very large BMO sets.

used_cars	1000	5000	10000	50000
BMO size	619	2931	5530	20047
$\mathbf{t_{rel}}$	2.3	44.2	159.3	2614.9
$\mathbf{t_{end}}$	1.9	39.1	139.1	2081.8
SF1	**1.2**	**1.1**	**1.1**	**1.3**

The preference transformations are:
`L7; L7; L9e; L1b; L1a`
Pushed and split preferences in T_{end}:

P_A = (BETWEEN(u.price, [5000, 30000]) ⊗ AROUND(u.age)) groupby {u.sid}

P_B = BETWEEN(u.price, [5000, 30000]) ⊗ AROUND(u.age) ⊗ BETWEEN(s.zipcode, [20000, 50000])

No specific evaluation algorithms can be applied here. The used general BNL algorithm degrades to $O(n^2)$, causing horrible performance figures in any case. ☼

We also executed some test queries on Preference SQL 1.3, running loosely coupled on Oracle 9i. SF2 as defined below compares it with our rapid prototype:

- Runtime $\mathbf{t_{Pref\text{-}SQL}}$ (in sec.)
- Performance speedup factor **SF2:=** $\mathbf{t_{Pref\text{-}SQL} / t_{end}}$

Though Preference SQL is known as reasonably fast, the subsequent numbers show already SF2 > 1, which is quite remarkable for a rapid prototype carrying this much overhead. (Note that SF2 could not be measured in each case, because prioritization P_1 & P_2 is supported in Preference SQL 1.3 only if P_1 is a chain.)

used_cars	1000	5000	10000	50000
SF2 : Q_2	2.5	1.4	1.5	3.5
SF2 : Q_3	2.0	3.0	2.0	2.3
SF2 : Q_4	2.0	2.0	2.5	2.0

5.3 Lessons Learned so far

So far we have seen cases with 1 < SF2 and SF1 being already quite good. But there are many more tuning opportunities. For instance, recall that we intentionally have favored join costs by implementing a fast hash join in XXL, but using the sub-optimal BNL algorithms for general preference evaluation. Clearly, if the latter is replaced by advances like [4, 19, 21, 22], SF1 will increase considerably. All specialized evaluation algorithms for base preference constructors were implemented by plain O(n) methods. Obviously the use of indexing achieves another substantial performance boost. When migrating to a tight integration the JDBC overhead will be eliminated either. Implementing all these add-on improvements is straightforward and will upgrade performance by large extents, maybe by orders of magnitude. Options for further performance speedup concern the sophistication of our Pass-2 and the topic of cost-based query optimization. Improving the quality of the 'push preference' heuristics is a learning process, similar to a classical SQL optimizers. Our algebraic approach to preference query optimization will be a good foundation to come up with cost estimation models for critical issues like preference selectivity.

In summary our experiments give already strong evidence that a tightly integrated preference query optimizer can achieve excellent performance. Enabled by the foundations of algebraic optimization we believe that we have revealed the entrance to a performance tuning gold mine, having many more options up our sleeves. A comprehensive performance testing is under way right now.

6 Related Works

The optimization of preference queries with BMO semantics poses new research challenges. Based on a technical report in July 2002 several preference relational algebra laws have been published by the authors in [15]. J. Chomicki's independent work focuses on a relaxation of strict partial order semantics for preferences: Our laws L1 and L2 are special cases of [6], L5 is a special case of [5]. Since most practical database applications seem to comply with strict partial order semantics, relaxing it must be carefully weighed; transformations can become invalidated, which in turn decreases preference query performance. Here we have presented a further series of new laws plus, for the first time, performance evaluations with a carefully engineered preference query optimizer.

In [4] a transformation for pushing skyline preferences through joins, being an instance of our law L6a, has been presented without a formal proof. Likewise a method for pushing skylines into a join, being an instance of law L7, can be found there. The notion of non-reductive joins is related to our laws L6 and L7.

BMO is compatible with the *top-k* query model, which has been quite popular with numerical preferences. A theorem in [5] ensures that in principle top-k can be provided by means of BMO as follows: If the result res of σ[P](R) has m tuples and m ≥ k, return k of them; otherwise deliver those m and find the remaining ones by σ[P](R \ res). In this way top-k query semantics can be provided for arbitrary strict partial order preferences, not only for numerical preferences. How skylines relate to top-k was addressed in [4] before.

Several algorithms have been proposed to efficiently implement numerical preferences for top-k querying, e.g. [1, 2, 7, 10, 11]. Efficient skyline algorithms were investigated e.g. by [4, 19, 21, 22].

7 Summary and Outlook

Assigning a strict partial order semantics to preferences is a good choice in many database applications. A rich repertoire of intuitive preference constructors can facilitate the design of personalized applications. Since numerical preferences are of limited expressiveness, constructors for Pareto and prioritized preferences are required too. One might argue that such a high modeling convenience leads to runtime inefficiency. However, our contributions give strong evidence that for relational preference queries with BMO semantics this is not the case.

We have laid the foundations of a framework for preference query optimization that extends established query optimization techniques from relational databases. Preference queries can be evaluated by preference relational algebra, extending classical relational algebra by two new preference operators. We have provided a series of novel transformation laws for preference relational algebra that are the key to algebraic optimization. A preference query optimizer can be constructed as an extension of existing SQL optimizers, adding new heuristics like 'push preference'. In this way the decade-long investments and experiences with relational query optimizer can be inherited completely. We presented a rapid prototype of such a preference query optimizer and carried out a series of performance experiments. The performance speedups observed so far give already strong evidence that a tightly coupled implementation inside an existing SQL query engine can achieve excellent performance.

Currently there are several projects within our "It's a Preference World" program that use Preference SQL or Preference XPATH, e.g. [8]. Building on the fundamental insights developed here for heuristic preference query optimization, the important issue of cost-based optimization can be tackled next. We will also extend our optimization framework to recursive preference queries, where we can start over from research results on pushing preference selection over recursion established already in [18]. Algebraic optimization of numerical preferences, which was not covered here, is another interesting topic.

In summary we believe that we have contributed a crucial stepping stone towards efficient preference query optimization: Database technology can offer very good support for preferences and personalization, both in terms of ease of modeling and high efficiency.

Acknowledgments

We thank W.T. Balke, J. Chomicki, U. Güntzer and Q. Wang for helpful comments on a draft of this paper.

References

[1] R. Agrawal, E. L. Wimmers: *A Framework for Expressing and Combining Preferences*. Proc. ACM SIGMOD, May 2000, Dallas, pp. 297 - 306.

[2] W.-T. Balke, U. Güntzer, W. Kießling: *On Real-time Top k Querying for Mobile Services*. Intern. Conf. on Cooperative Information Systems (CoopIS), Irvine, CA, USA, Nov. 2002, pp. 125-143.

[3] J. Bercken, B. Blohsfeld, J. Dittrich, et al.: *XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries*. Proc. 27th VLDB, Rome, Sept. 2001, pp. 39-48.

[4] S. Börzsönyi, D. Kossmann, K. Stocker: *The Skyline Operator*. Proc. 17th Intern. Conf. On Data Engineering (ICDE), Heidelberg, April 2001, pp. 421-430.

[5] J. Chomicki: *Querying with Intrinsic Preferences*. Proc. Intern. Conf. on Advances in Database Technology (EDBT), Prague, March 2002, pp. 34-51.

[6] J. Chomicki*: Preference Queries in Relational Databases*. Univ. of Buffalo, Online Technical Report, arXiv:cs.DB/0207093, Aug 2002.

[7] R. Fagin, A. Lotem, M. Naor: *Optimal Aggregation Algorithms for Middleware*. Proc. ACM PODS, Santa Barbara, May 2001, pp. 102-113.

[8] S. Fischer, W. Kießling, S. Holland, M. Fleder: *The COSIMA Prototype for Multi-Objective Bargaining,* 1st Intern. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS), Bologna, July 2002, pp. 1364-1371.

[9] P. C. Fishburn: *Preference Structures and their Numerical Representations*. Theoretical Computer Science, 1999, 217:359-383.

[10] U. Güntzer, W.-T. Balke, W.Kießling: *Optimizing Multi-Feature Queries for Image Databases*. Proc. 26th VLDB, Cairo, Egypt, Sept. 2000, pp. 419-428.

[11] V. Hristidis, N. Koudas, Y. Papakonstantinou: *PREFER : A System for the Efficient Execution of Multiparametric Ranked Queries*. Proc. ACM SIGMOD, May 2001, Santa Barbara, pp. 259 - 269.

[12] R. Keeney, H. Raiffa: *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, UK, 1993.

[13] W. Kießling: *Foundations of Preferences in Database Systems.* Proc. 28th VLDB, Hong Kong, China, Aug. 2002, pp. 311-322.

[14] W. Kießling, U. Güntzer: *Database Reasoning - A Deductive Framework for Solving Large and Complex Problems by means of Subsumption*. Proc. 3rd Workshop on Information Systems and Artificial Intelligence, LNCS 777, Hamburg, 1994, pp. 118-138.

[15] W. Kießling, B. Hafenrichter: *Optimizing Preference Queries for Personalized Web Services*. IASTED Intern. Conf. on Comm., Internet, Information Technology. St. Thomas, USA, Nov. 2002, pp. 461 - 466.

[16] W. Kießling, B. Hafenrichter, S. Fischer, S. Holland: *Preference XPATH: A Query Language for E-Commerce*. Proc. 5th Intern. Konf. für Wirtschaftsinformatik, Augsburg , Sept. 2001, pp. 425-440.

[17] W. Kießling, G. Köstler: *Preference SQL – Design, Implementation, Experiences*. Proc. 28th VLDB, Hong Kong, China, Aug. 2002, pp. 990-1001.

[18] G. Köstler, W. Kießling, H. Thöne, U. Güntzer: *Fixpoint Iteration with Subsumption in Deductive Databases*. Journal of Intelligent Information Systems, Vol. 4, Boston, USA, 1995, pp. 123-148.

[19] D. Kossmann, F. Ramsak, S. Rost: *Shooting Stars in the Sky: An Online Algorithm for Skyline Queries.* Proc. 28th VLDB, Hong Kong, Aug. 2002.

[20] M. Lacroix, P. Lavency : *Preferences : Putting More Knowledge into Queries.* Proc. 13th VLDB, Brighton, 1987, pp. 217-225.

[21] D. Papadias, Y. Tao, G. Fu, B. Seeger: *An Optimal and Progressive Algorithm for Skyline Queries.* Proc. ACM SIGMOD*,* June 2003, San Diego.

[22] K.-L. Tan, P.-K. Eng, B. C. Ooi: *Efficient Progressive Skyline Computation*. Proc. 27th VLDB, Rome, Sept. 2001, pp. 301-310.

[23] J. Ullman: *Principles of Database and Knowledge-Base Systems*. Vol. 1, Comp. Science Press, 1989.

Appendix 1: Proofs

All proofs use the definitions of a preference $P = (A, <_P)$, preference selection $\sigma[P](R)$ and grouped preference selection $\sigma[P \text{ groupby } B](R)$ as stated in section 2.

Lemma 1: (proof obvious)

Given $P = (A, <_P)$ and $A \subseteq X \subseteq \text{attr}(R)$. Then P induces a strict partial order $<_X$ onto R:

$$v, w \in R:\ v <_X w \iff v[A] <_P w[A]$$

Lemma 2:

Let f(r, s) be a left-total Boolean function, i.e. for every r there exists at least one s satisfying f(r, s). The *extension of R by S through f* is defined as:

$$R \text{ ext}_f\, S = \{(r, s) \mid r \in R, s \in S, f(r, s)\}$$

Let $P = (A, <_P)$ and $A \subseteq \text{attr}(R)$, then:

$$\sigma[P](R \text{ ext}_f\, S) = \sigma[P](R) \text{ ext}_f\, S$$

Proof:

$\sigma[P](R \text{ ext}_f\, S)$

$= \{w \in R \text{ ext}_f\, S \mid \neg\exists v \in R \text{ ext}_f\, S: w[A] <_P v[A]\}$

$= \{w \in R \text{ ext}_f\, S \mid \neg\exists v \in R : w[A] <_P v[A]\}$

$= \{w \in R \mid \neg\exists v \in R : w[A] <_P v[A]\} \text{ ext}_f\, S$

$= \sigma[P](R) \text{ ext}_f\, S$ ♦

Lemma 3:

Let $P_1 = (A, <_{P1})$, $P_2 = (A, <_{P2})$ and $\sigma[P_1](R) \subseteq \sigma[P_2](R)$:

$$\sigma[P_1](R) = \sigma[P_1](\sigma[P_2](R))$$

Proof:

$\sigma[P_1](R) = \sigma[P_1](R) \cap \sigma[P_2](R)$

$= \{w \in R \mid \neg\exists v \in R: w[A] <_{P1} v[A]\} \cap \sigma[P_2](R)$

// Due to $\sigma[P_1](R) \subseteq \sigma[P_2](R)$ we have:

// $\neg\exists v \in R: w[A] <_{P1} v[A]$ implies

// $\neg\exists v \in \sigma[P_2](R): w[A] <_{P1} v[A]$

$= \{w \in R \mid \neg\exists v \in \sigma[P_2](R): w[A] <_{P1} v[A]\} \cap \sigma[P_2](R)$

$= \{w \in \sigma[P_2](R) \mid \neg\exists v \in \sigma[P_2](R): w[A] <_{P1} v[A]\}$

$= \sigma[P_1](\sigma[P_2](R))$ ♦

Lemma 4:

$$\sigma[P_1 \otimes P_2](R) \subseteq \sigma[P_1 \text{ groupby } A_2](R)$$

Proof : Let $P = (A, <_P) := (A_1 \cup A_2, <_{P1 \otimes P2})$.

$\sigma[P_1 \otimes P_2](R)$

$= \{w \in R \mid \neg\exists v \in R : w[A] <_P v[A]\}$ // Def '⊗'

$= \{w \in R \mid \neg(\exists v \in R:$
$(w[A_1] <_{P1} v[A_1] \wedge$
$(w[A_2] = v[A_2] \vee w[A_2] <_{P2} v[A_2])) \vee$
$(w[A_2] <_{P2} v[A_2] \wedge$
$(w[A_1] = v[A_1] \vee w[A_1] <_{P1} v[A_1])))\}$

$= \{w \in R \mid \neg(\exists v \in R:$
$(w[A_1] <_{P1} v[A_1] \wedge w[A_2] = v[A_2]) \vee$
$(w[A_1] <_{P1} v[A_1] \wedge w[A_2] <_{P2} v[A_2]) \vee$
$(w[A_2] <_{P2} v[A_2] \wedge w[A_1] = v[A_1]))\}$

$= \{w \in R \mid$
$\neg((\exists v \in R: (w[A_1] <_{P1} v[A_1] \wedge w[A_2] = v[A_2]))$
$\vee (\exists v \in R: (w[A_1] <_{P1} v[A_1] \wedge w[A_2] <_{P2} v[A_2]))$
$\vee (\exists v \in R: (w[A_2] <_{P2} v[A_2] \wedge w[A_1] = v[A_1])))\}$

$= \{w \in R \mid$
$(\neg\exists v \in R: (w[A_1] <_{P1} v[A_1] \wedge w[A_2] = v[A_2]))$
$\wedge (\neg\exists v \in R: (w[A_1] <_{P1} v[A_1] \wedge w[A_2] <_{P2} v[A_2]))$
$\wedge (\neg\exists v \in R: (w[A_2] <_{P2} v[A_2] \wedge w[A_1] = v[A_1]))\}$

$= \sigma[P_1 \text{ groupby } A_2](R) \cap \sigma[P_2 \text{ groupby } A_1](R) \cap$
$\{w \in R \mid \exists v \in R: (w[A_1] <_{P1} v[A_1] \wedge w[A_2] <_{P2} v[A_2])\}$

$\subseteq \sigma[P_1 \text{ groupby } A_2](R)$ ♦

Lemma 5:

Let $P=(A,<_P)$, $\text{attrib}(P) \subseteq R$, $X \subseteq \text{attribs}(R) \cap \text{attirb}(S)$ then

$$\sigma[P \text{ groupby } X](R \ltimes_{R.X = S.X} S) \bowtie_{R.X = S.X} S = \sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S$$

Proof:

$T' = R \ltimes_{R.X = S.X} S$

$\sigma[P \text{ groupby } X](R \ltimes_{R.X = S.X} S)$

$= \{ w \in T' \mid \neg\exists v \in T' : w < v \wedge w[x] = v[x] \}$

$= \{ w \in R \mid \neg\exists v \in R: w < v \wedge w[x] = v[x]$
$\wedge w \in T' \wedge v \in T' \}$

// w[x]=v[x] , transitivity

$= \{ w \in R \mid \neg\exists v : w < v \wedge w[x] = v[x] \wedge w \in T'\}$

$= \{ w \in R \mid \neg\exists v : w < v \wedge w[x] = v[x] \} \cap T'$

$= \sigma[P \text{ groupby } X](R) \cap R \ltimes_{R.X = S.X} S$

$\sigma[P \text{ groupby } X](R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S$
$= (\sigma[P \text{ groupby } X](R) \cap R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S$
$= (\sigma[P \text{ groupby } X](R) \bowtie_{R.X=S.X} S)$
$\qquad \cap ((R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S)$
$= \sigma[P \text{ groupby } X](R) \bowtie_{R.X=S.X} S \cap (R \bowtie_{R.X=S.X} S)$
$= \sigma[P \text{ groupby } X](R) \bowtie_{R.X=S.X} S$ ♦

Theorem L1: *Push projection over preference*

Let $P = (A, <_P)$ and $A, X \subseteq attr(R)$.

a) $\sigma[P](\pi_X(R)) = \pi_X(\sigma[P](R))$ if $A \subseteq X$
b) $\pi_X(\sigma[P](\pi_{X \cup A}(R))) = \pi_X(\sigma[P](R))$ otherwise

Proof: a) Given $P = (A,<_P)$, $A \subseteq X \subseteq attr(R)$, then:
$\pi_X(\sigma[P](R))$
$= \pi_X(\{w \in R \mid \neg\exists v \in R : w[A] <_P v[A]\})$ // Lem1
$= \pi_X(\{w \in R \mid \neg\exists v \in R : w <_X v\})$
$= \{w \in \pi_X(R) \mid \neg\exists v \in \pi_X(R) : w <_X v\}$ // Lem1
$= \sigma[P](\pi_X(R))$ ♦

Proof: b) Given $P = (A,<_P)$ and $\neg(A \subseteq X)$, then:
$\pi_X(\sigma[P](R)) = \pi_X(\pi_{X \cup A}(\sigma[P](R)))$ // ThmL1a
$= \pi_X(\sigma[P](\pi_{X \cup A}(R))$ ♦

Theorem L2: *Push preference over Cartesian product*

Let $P = (A, <_P)$ and $A \subseteq attr(R)$.
$\sigma[P](R \times S) = \sigma[P](R) \times S$

Proof: $R \times S$ can be rewritten in terms of an extension function: $R \times S = R \text{ ext}_\times S, \times(r, s) = true$
$\sigma[P](R \times S)$
$= \sigma[P](R \text{ ext}_\times S)$ // Lem2
$= \sigma[P](R) \text{ ext}_\times S = \sigma[P](R) \times S$ ♦

Theorem L3: *Push preference over union*
Let $P = (A, <_P)$ and $A \subseteq attr(R) = attr(S)$.
$\sigma[P](R \cup S) = \sigma[P](\sigma[P](R) \cup \sigma[P](S))$

Proof: Let $T := \sigma[P](R) \cup \sigma[P](S)$.
$\sigma[P](T)$
$= \{w \in T \mid \neg\exists v \in T : w[A] <_P v[A]\}$ // T
$= \{w \in T \mid \neg((\exists v \in \sigma[P](R) : w[A] <_P v[A]) \vee$
$\qquad (\exists v \in \sigma[P](S) : w[A] <_P v[A]))\}$
$= \{w \in T \mid \neg((\exists v \in R : w[A] <_P v[A] \wedge v \in \sigma[P](R)) \vee$
$\qquad (\exists v \in S : w[A] <_P v[A] \wedge v \in \sigma[P](S)))\}$
// since $<_P$ is transitive
$= \{w \in T \mid \neg((\exists v \in R : w[A] <_P v[A]) \vee$
$\qquad (\exists v \in S : w[A] <_P v[A]))\}$
$= \{w \in R \cup S \mid \neg(\exists v \in R \cup S: w[A] <_P v[A]) \wedge w \in T\}$
// again by transitivity of $<_P$
$= \{w \in R \cup S \mid \neg(\exists v \in R \cup S : w[A] <_P v[A])\}$
$= \sigma[P](R \cup S)$ ♦

Theorem L4: *Split prioritization into grouping*
Let $P_1 = (A_1, <_{P1})$, $P_2 = (A_2, <_{P2})$ and $A_1, A_2 \subseteq attr(R)$.
$\sigma[P_1 \,\&\, P_2](R) = \sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R))$

Proof:
$\sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R))$
$= \{w \in \sigma[P_1](R) \mid \neg\exists v \in \sigma[P_1](R) :$
$\qquad w[A_1] = v[A_1] \wedge w[A_2] <_{P2} v[A_2] \}$
$= \{w \in R \mid \neg\exists v \in R: w[A_1] = v[A_1] \wedge w[A_2] <_{P2} v[A_2]$
$\qquad \wedge v \in \sigma[P_1](R) \wedge w \in \sigma[P_1](R)\}$
// $w \in \sigma[P_1](R)$, $w[A_1] = v[A_1]$ implies $v \in \sigma[P_1](R)$
$= \{w \in R \mid \neg\exists v \in R: w[A_1] = v[A_1] \wedge w[A_2] <_{P2} v[A_2]$
$\qquad \wedge w \in \sigma[P_1](R)\}$
$= \{w \in R \mid \neg\exists v \in R: w[A_1] = v[A_1] \wedge w[A_2] <_{P2} v[A_2]\}$
$\quad \cap \sigma[P_1](R)\}$
$= \sigma[P_1](R) \cap \sigma[P_2 \text{ groupby } A_1](R) = \sigma[P_1 \,\&\, P_2](R)$
The last equality has been given already in [13]. ♦

Theorem L5: Push preference over hard selection

$\sigma[P](\sigma_H(R)) = \sigma_H(\sigma[P](R))$ iff
$\forall w \in R: (H(w) \wedge \exists v \in R: w[A] <_P v[A] \text{ implies } H(v))$

Proof: We first prove an *auxiliary lemma*:
$\sigma_H(\sigma[P](R)) \subseteq \sigma[P](\sigma_H(R))$

Proof of this auxiliary lemma:
$w \in \sigma_H(\sigma[P](R))$
iff $\neg(\exists v \in R: w[A] <_P v[A]) \wedge H(w)$
iff $\neg(\exists v \in R: w[A] <_P v[A] \wedge H(w))$ // *1*

On the other hand we have:
$w \in \sigma[P](\sigma_H(R))$
iff $w \in \sigma_H(R) \wedge H(w)$
iff $\neg(\exists v \in R: w[A] <_P v[A] \wedge H(v)) \wedge H(w)$
iff $\neg(\exists v \in R: w[A] <_P v[A] \wedge H(v) \wedge H(w))$ // *2*
Now we can conclude:
$\sigma_H(\sigma[P](R)) \subseteq \sigma[P](\sigma_H(R))$
iff $\forall w \in R$: *1* implies *2*
iff $\forall w \in R : (\exists v \in R: w[A] <_P v[A] \wedge H(v) \wedge H(w)$
$\qquad$ implies $\exists v \in R: w[A] <_P v[A] \wedge H(w))$ iff true

The reverse set inclusion, which we also require to prove theorem L5, holds only for the stated non-trivial condition. We start over from lines labeled *1* and *2* above:
$w \in \sigma_H(\sigma[P](R))$
iff $\neg(\exists v \in R: w[A] <_P v[A] \wedge H(w))$
$w \in \sigma[P](\sigma_H(R))$
iff $\neg(\exists v \in R: w[A] <_P v[A] \wedge H(v) \wedge H(w))$
Now we can conclude:
$w \in \sigma[P](\sigma_H(R))$ implies $w \in \sigma_H(\sigma[P](R))$
iff $\exists v \in R: w[A] <_P v[A] \wedge H(w)$ implies
$\qquad \exists v \in R: w[A] <_P v[A] \wedge H(v) \wedge H(w)$
iff $\exists v \in R: w[A] <_P v[A] \wedge H(w)$ implies $H(v)$
iff $H(w) \wedge \exists v \in R: w[A] <_P v[A]$ implies $H(v)$ ♦

Corollary 2: *Special cases of L5*

Let P_1 := LOWEST(A), P_2 := HIGHEST(A).

a. $\sigma[P_1](\sigma_{A <= c}(R)) = \sigma_{A <= c}(\sigma[P_1](R))$

b) $\sigma[P_2](\sigma_{A >= c}(R)) = \sigma_{A >= c}(\sigma[P_2](R))$

Proof : a) Directly from Theorem L5

$\forall w \in R: (H(w) \wedge \exists\, v \in R: w[A] <_P v[A] \rightarrow H(v))$

// Def $H(w) = A \leq c$
iff $\forall w \in R: (w[A] \leq c \wedge \exists\, v \in R: w[A] <_P v[A]$ implies $v[A] \leq c)$
// Def 'LOWEST'
iff $\forall w \in R: (w[A] \leq c \wedge \exists\, v \in R: w[A] > v[A]$ implies $v[A] \leq c)$
// Transitivity: $v[A] < w[A] \leq c$
iff true

Hence $\sigma[P_1](\sigma_{A <= c}(R)) = \sigma_{A <= c}(\sigma[P_1](R))$ holds. ♦

Proof : b) Directly from Theorem L5

$\forall w \in R: (H(w) \wedge \exists\, v \in R: w[A] <_P v[A] \rightarrow H(v))$

// Def $H(w) = A \geq c$
iff $\forall w \in R: (w[A] \geq c \wedge \exists\, v \in R: w[A] <_P v[A]$ implies $v[A] \geq c)$
// Def 'HIGHEST'
iff $\forall w \in R: (w[A] \geq c \wedge \exists\, v \in R: w[A] < v[A]$ implies $v[A] \geq c)$
// Transitivity: $v[A] > w[A] \geq c$
iff true

Hence, $\sigma[P_2](\sigma_{A >= c}(R)) = \sigma_{A >= c}(\sigma[P_2](R))$ holds. ♦

Theorem L6: Push preference over a join

Let $P = (A, <_P)$, $A \subseteq attr(R)$ and $X \subseteq attr(R) \cap attr(S)$.

a) $\sigma[P](R \bowtie_{R.X = S.X} S) = \sigma[P](R) \bowtie_{R.X = S.X} S$,
if each tuple in R has at least one join partner in S

Let $R \ltimes_{R.X = S.X} S$ denote a semi-join operation.

b) $\sigma[P](R \bowtie_{R.X = S.X} S) =$
$\sigma[P](R \ltimes_{R.X = S.X} S) \bowtie_{R.X = S.X} S$

c) $\sigma[P](R \bowtie_{R.X = S.X} S)) =$
$\sigma[P](\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S)$

Further let $B \subseteq attr(S)$.

d) $\sigma[P \text{ groupby } B](R \bowtie_{R.X = S.X} S) =$
$\sigma[P \text{ groupby } B](\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S))$

$R \bowtie_{R.X = S.X} S = \sigma_{R.X = S.X}(R \times S) = R\ ext_{join}\ S$
join(r, s) = {true : r[X] = s[X], false otherwise}

Proof: a) Due to the assumption, each tuple in R has at least one join partner in S, the join can be rewritten in terms of an extension function:

$\sigma[P](R \bowtie_{R.X = S.X} S)$
$= \sigma[P](R\ ext_{join}\ S)$ // Lem2
$= \sigma[P](R)\ ext_{join}\ S$
$= \sigma[P](R) \bowtie_{R.X = S.X} S$ ♦

Proof: b)

$\sigma[P](R \bowtie_{R.X=T.X} S)$
$= \sigma[P]((R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S)$ //Lem2
$= \sigma[P]((R \ltimes_{R.X=S.X} S)\ ext_{join}\ S)$
//Lem2, $\forall r \in R: \exists s \in S: r[X] = s[X]$
$= \sigma[P](R \ltimes_{R.X=S.X} S))\ ext_{join}\ S$ //Lem2
$= \sigma[P](R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S$ ♦

Proof: c)

$\sigma[P](R \bowtie_{R.X = S.X} S)$ //Lem3
$= \sigma[P](\sigma[P \text{ groupby } R.X](R \bowtie_{R.X = S.X} S))$ //ThmL7b
$= \sigma[P](\sigma[P \text{ groupby } R.X](R \ltimes_{R.X = S.X} S)$
$\bowtie_{R.X = S.X} S)$ //Lem5
$= \sigma[P](\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S)$ ♦

Proof: d)
$\sigma[P \text{ groupby } B](R \bowtie_{R.X = S.X} S)$ // $S = \{s_1,..,s_n\}$
$= \sigma[P \text{ groupby } B]\ (R \bowtie_{R.X = S.X} \{s_1\} \cup ... \cup$
$R \bowtie_{R.X = S.X} \{s_n\})$ // ThmL3
$= \sigma[P \text{ groupby } B]\ ($
$\sigma[P \text{ groupby } B](R \bowtie_{R.X = S.X} \{s_1\}) \cup ... \cup$
$\sigma[P \text{ groupby } B](R \bowtie_{R.X = S.X} \{s_n\}))$
// Since $B \subseteq attr(S)$: $\forall v \in R \bowtie_{R.X = S.X} \{s_i\}$: $v[B] = s_i[B]$
$= \sigma[P \text{ groupby } B]\ (\sigma[P](R \bowtie_{R.X = S.X} \{s_1\})$
$\cup ... \cup \sigma[P](R \bowtie_{R.X = S.X} \{s_n\}))$ // ThmL6c
$= \sigma[P \text{ groupby } B]\ ($
$\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} \{s_1\} \cup ... \cup$
$\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} \{s_n\})$
$= \sigma[P \text{ groupby } B](\sigma[P \text{ groupby } X](R) \bowtie_{R.X = S.X} S))$ ♦

Theorem L7: *Split Pareto preference and push over join*

Let $P_1 = (A_1, <_{P1})$ where $A_1 \subseteq attr(R)$, $P_2 = (A_2, <_{P2})$ where $A_2 \subseteq attr(S)$. Further let $X \subseteq attr(R) \cap attr(S)$.

$\sigma[P_1 \otimes P_2](R \bowtie_{R.X = S.X} S) =$
$\sigma[P_1 \otimes P_2]\ (\sigma[P_1 \text{ groupby } X](R) \bowtie_{R.X = S.X} S)$

Proof: Let $T := R \bowtie_{R.X = S.X} S$.
// Lem 4
$\sigma[P_1 \otimes P_2](T) \subseteq \sigma[P_1 \text{ groupby } A_2](T)$ // Lem3
$\sigma[P_1 \otimes P_2](R \bowtie_{R.X = S.X} S)$
$= \sigma[P_1 \otimes P_2](\sigma[P_1 \text{ groupby } A_2](R \bowtie_{R.X = S.X} S))$
//ThmL6d
$= \sigma[P_1 \otimes P_2](\sigma[P_1 \text{ groupby } A_2]$

$(\sigma[P_1 \text{ groupby } R.X](R) \bowtie_{R.X=S.X} S))$ // Lem3
$= \sigma[P_1 \otimes P_2](\sigma[P_1 \text{ groupby } R.X](R) \bowtie_{R.X=S.X} S)$ ♦

Theorem L8: *Split prioritization and push over join*

Let $P_1 = (A_1, <_{P1})$ where $A_1 \subseteq \text{attr}(R)$, $P_2 = (A_2, <_{P2})$ where $A_2 \subseteq \text{attr}(R) \cup \text{attr}(S)$ and let $X \subseteq \text{attr}(R) \cap \text{attr}(S)$.

a) $\sigma[P_1 \,\&\, P_2](R \bowtie_{R.X=S.X} S) =$
$\sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R) \bowtie_{R.X=S.X} S)$,
if each tuple in R has at least one join partner in S

b) $\sigma[P_1 \,\&\, P_2](R \bowtie_{R.X=S.X} S) =$
$\sigma[P2 \text{ groupby } A_1](\sigma[P_1](R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S)$

Proof: a)
$\sigma[P_1 \,\&\, P_2](R \bowtie_{R.X=S.X} S)$ // ThmL4
$= \sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R \bowtie_{R.X=S.X} S))$ // ThmL6a
$= \sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R) \bowtie_{R.X=S.X} S)$ ♦

Proof: b)
$\sigma[P_1 \,\&\, P_2](R \bowtie_{R.X=S.X} S)$ // ThmL4
$= \sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R \bowtie_{R.X=S.X} S))$ // ThmL6c
$= \sigma[P_2 \text{ groupby } A_1](\sigma[P_1](R \ltimes_{R.X=S.X} S) \bowtie_{R.X=S.X} S)$ ♦

Theorem L9: *Simplify grouped preference selections*

a) $\sigma[(P \text{ groupby } B_1) \text{ groupby } B_2](R) =$
$\sigma[P \text{ groupby } (B_1 \cup B_2)](R)$

b) $\sigma[(P_1 \text{ groupby } B) \,\&\, P_2](R) =$
$\sigma[(P_1 \,\&\, P_2) \text{ groupby } B](R)$

c) $\sigma[P_1 \otimes (P_2 \text{ groupby } B)](R) =$
$\sigma[(P_1 \otimes P_2) \text{ groupby } B](R)$

d) $\sigma[(P_1 \text{ groupby } B_1) \otimes (P_2 \text{ groupby } B_2)](R) =$
$\sigma[(P_1 \otimes P_2) \text{ groupby } (B_1 \cup B_2)](R)$

e) $\sigma[P \text{ groupby } B](R) = R$ if B is key in R

Proof: a)
$\sigma[((P \text{ groupby } B_1) \text{ groupby } B_2)$
$= \sigma[B_2^{\leftrightarrow} \,\&\, (B_1^{\leftrightarrow} \,\&\, P)](R)$ // ThmL11c
$= \sigma[(B_2^{\leftrightarrow} \,\&\, B_1^{\leftrightarrow}) \,\&\, P)](R)$ // Def '&'
$= \sigma[(B_2 \cup B_1)^{\leftrightarrow} \,\&\, P)](R)$
$= \sigma[P \text{ groupby } (B_1 \cup B_2)](R)$ ♦

Proof: b)
$\sigma[(P_1 \text{ groupby } B) \,\&\, P_2](R)$
$= \sigma[(B^{\leftrightarrow} \,\&\, P_1) \,\&\, P_2](R)$ // ThmL11c
$= \sigma[B^{\leftrightarrow} \,\&\, (P_1 \,\&\, P_2)\,](R)$
$= \sigma[(P_1 \,\&\, P_2) \text{ groupby } B](R)$ ♦

Proof: c)
$\sigma[P_1 \otimes (P_2 \text{ groupby } B)](R)$ // (**)
$= \sigma[P_1 \otimes (P_2 \otimes B^{\leftrightarrow})](R)$ // ThmL11b
$= \sigma[(P_1 \otimes P_2) \otimes B^{\leftrightarrow}](R)$ // (**)
$= \sigma[(P_1 \otimes P_2) \text{ groupby } B](R)$
Note that (**) holds due to proposition 3k in [13]. ♦

Proof: d)
$\sigma[(P_1 \text{ groupby } B_1) \otimes (P_2 \text{ groupby } B_2)](R)$
$= \sigma[(P_1 \otimes B_1^{\leftrightarrow}) \otimes (P_2 \otimes B_2^{\leftrightarrow})](R)$ // ThmL11a,b
$= \sigma[((P_1 \otimes P_2) \otimes B_2^{\leftrightarrow}) \otimes B_1^{\leftrightarrow}](R)$ // (**)
$= \sigma[((P_1 \otimes P_2) \text{ groupby } B_2) \otimes B_1^{\leftrightarrow}](R)$ // (**)
$= \sigma[((P_1 \otimes P_2) \text{ groupby } B_2) \text{ groupby } B_1](R)$ // ThmL9a
$= \sigma[(P_1 \otimes P_2) \text{ groupby } (B_1 \cup B_2)](R)$ ♦

Proof: e)

$\sigma[P \text{ groupby } X](R)$
$=\{\, w \in R \mid \neg\exists v \in R : w[X] = v[X] \wedge w <_P v \,\}$
// $w = v$ $\quad w <_P v = \text{false}$
// $w \neq v$ $\quad w[X] = v[X] = \text{false}$, X is unique
$=\{\, w \in R \mid \neg\exists v \in R : \text{false} \,\}$
$= R$ ♦

Theorem L10: *Cascade of preference and hard selection*

a) Let $P = (A, <_P)$ and $A, B \subseteq \text{attr}(R)$:
$\sigma[P](\sigma_{B=c}(R)) = \sigma_{B=c}(\sigma[P \text{ groupby } B](R))$
$\sigma[P](\sigma_{A=c}(R)) = \sigma_{A=c}(R)$

b) Let $P := \text{BETWEEN}(A, [c_1, c_2])$:
$\sigma[P](\sigma_{A>=c}(R)) = \sigma[\text{LOWEST}(A)](\sigma_{A>=c}(R))$, if $c > c_2$
$\sigma[P](\sigma_{A<=c}(R)) = \sigma[\text{HIGHEST}(A)](\sigma_{A<=c}(R))$, if $c < c_1$
$\sigma[P](\sigma_{A>=c1 \wedge A<=c2}(R)) = \sigma_{A>=c1 \wedge A<=c2}(R)$

Let $H :=$ '$A = c_1 \vee \ldots \vee A = c_n$', H-set $:= \{c_1, \ldots, c_n\}$.

c) Let $P := \text{POS}(A, \text{Pos-set})$, Pos-set $= \{v_1, \ldots, v_m\}$:
if H-set $\cap$ Pos-set $= \emptyset$ or H-set $\subseteq$ Pos-set
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

d) Let $P := \text{NEG}(A, \text{Neg-set})$, Neg-set $= \{v_1, \ldots, v_m\}$:
if H-set $\cap$ Neg-set $= \emptyset$ or H-set $\subseteq$ Neg-set
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

e) Let $P = \text{POS/NEG}(A, \text{POS-set}, \text{NEG-set})$,
POS-set$=\{v_1, \ldots, v_m\}$, NEG-set $= \{v_{m+1}, \ldots, v_{m+n}\}$:
If H-set $\subseteq$ POS-set or H-set $\subseteq$ NEG-set or
H-set $\cap$ (POS-Set $\cup$ NEG-Set) $= \emptyset$
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

f) Let $P = \text{POS/POS}(A, \text{POS1-set}, \text{POS2-set})$,
POS1-set$=\{v_1, \ldots, v_m\}$, POS2-set $= \{v_{m+1}, \ldots, v_{m+n}\}$:
If H-set $\subseteq$ POS1-set or H-set $\subseteq$ POS2-set or
H-set $\cap$ (POS1-Set $\cup$ POS2-Set) $= \emptyset$
then $\sigma[P](\sigma_H(R)) = \sigma_H(R)$

Proof: a)

$\sigma[P](\sigma_{B=c}(R))$
$= \{w \in \sigma_{B=c}(R) \mid \neg\exists v \in \sigma_{B=c}(R) : w[A] <_P v[A]\}$
$= \{w \in R \mid \neg\exists v \in \sigma_{B=c}(R) : w[A] <_P v[A] \wedge w[B] = c\}$
$= \{w \in R \mid \neg\exists v \in R :$
$\quad w[A] <_P v[A] \wedge w[B] = c \wedge v[B] = c\}$
$= \{w \in R \mid \neg\exists v \in R :$
$\quad w[A] <_P v[A] \wedge v[B] = w[B] \wedge w[B] = c\}$
$= \{w \in \sigma[P \text{ groupby } B](R) \mid w[B] = c\}$
$= \sigma_{B=c}(\sigma[P \text{ groupby } B](R))$ ♦

$\sigma[P](\sigma_{A=c}(R))$
$= \{w \in \sigma_{A=c}(R) \mid \neg\exists v \in \sigma_{A=c}(R): w[A] <_P v[A]\}$
// $w[A] = v[A] = c$
$= \{w \in \sigma_{A=c}(R) \mid \neg\exists v \in \sigma_{A=c}(R): \text{false}\}$
$= \sigma_{A=c}(R)$ ♦

Proof: b)

Let $T := \sigma_{A>=c}(R)$, $P :=$ LOWEST(A).
$\sigma[P](T)$
$= \{w \in T \mid \neg\exists v \in T: w[A] >_P v[A]\}$
// Def. 'LOWEST', assumption $c > c_2$
$= \{w \in T \mid \neg\exists v \in T: w[A] - c_2 > v[A] - c_2\}$
// Def 'BETWEEN', assumption $c > c_2$
$= \sigma[\text{BETWEEN}(A, [c_1, c_2])](T)$ ♦

Let $T := \sigma_{A<=c}(R)$, $P :=$ HIGHEST(A).
$\sigma[P](T)$
$= \{w \in T \mid \neg\exists v \in T: w[A] <_P v[A]\}$
// Def. 'HIGHEST'
$= \{w \in T \mid \neg\exists v \in T: -w[A] > -v[A]\}$
// assumption $c < c_1$
$= \{w \in T \mid \neg\exists v \in T: c_1 - w[A] > c_1 - v[A]\}$
// Def 'BETWEEN', assumption $c < c_1$
$= \sigma[\text{BETWEEN}(A, [c_1, c_2])](T)$ ♦

Let $T := \sigma_{A>=c1 \wedge A<=c2}(R)$.
$\sigma[\text{BETWEEN}(A, [c_1, c_2])](T)$ // Def 'BETWEEN'
$= \{w \in T \mid \neg\exists v \in T: \text{distance}(w) > \text{distance}(v)\}$
$= \{w \in T \mid \neg\exists v \in T: 0 > 0\}$
$= \{w \in T \mid \text{true}\} = T$ ♦

Proof: c)

Immediately from Theorem L10e, since POS is a subconstructor of POS/POS.

P:=POS(A, POS-set)=POS/POS(A, POS-set, Ø) ♦

Proof: d)

Immediately from Theorem L10f, since NEG is a subconstructor of POS/NEG.

P:=NEG(A, NEG-set)=POS/NEG(A, Ø, NEG-set) ♦

Proof: e)

H-set ⊆ POS-set

$\sigma[P](\sigma_H(R))$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : w <_P v\}$ //Def. POS/NEG
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(w[A] \in \text{NEG-set} \wedge v[A] \notin \text{NEG-SET})$
$\vee\ (w[A] \notin \text{NEG-set} \wedge w[A] \notin \text{POS-set}$
$\wedge\ v[A] \in \text{POS-set})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(\text{false} \wedge \text{true}) \vee (\text{true} \wedge \text{false} \wedge \text{true})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : \text{false}\}$
$= \sigma_H(R)$ ♦

H-set ⊆ NEG-set

$\sigma[P](\sigma_H(R))$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : w <_P v\}$ //Def. POS/NEG
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(w[A] \in \text{NEG-set} \wedge v[A] \notin \text{NEG-SET})$
$\vee\ (w[A] \notin \text{NEG-set} \wedge w[A] \notin \text{POS-set}$
$\wedge\ v[A] \in \text{POS-set})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(\text{true} \wedge \text{false}) \vee (\text{false} \wedge \text{true} \wedge \text{false})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : \text{false}\}$
$= \sigma_H(R)$ ♦

H-set ∩ (POS-Set ∪ NEG-Set) = Ø

$\sigma[P](\sigma_H(R))$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : w <_P v\}$ //Def. POS/NEG
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(w[A] \in \text{NEG-set} \wedge v[A] \notin \text{NEG-SET})$
$\vee\ (w[A] \notin \text{NEG-set} \wedge w[A] \notin \text{POS-set}$
$\wedge\ v[A] \in \text{POS-set})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(\text{false} \wedge \text{true}) \vee (\text{true} \wedge \text{true} \wedge \text{false})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : \text{false}\}$
$= \sigma_H(R)$ ♦

Proof: f)

H-set ⊆ POS1-set

$\sigma[P](\sigma_H(R))$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : w <_P v\}$ //Def. POS/POS
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(w[A] \in \text{POS2-set} \wedge v[A] \in \text{POS1-set})$
$\vee\ (w[A] \notin \text{POS1-set} \wedge w[A] \notin \text{POS2-set}$
$\wedge\ v[A] \in \text{POS2-set})$
$\vee\ (w[A] \notin \text{POS1-set} \wedge w[A] \notin \text{POS2-set}$
$\wedge\ v[A] \in \text{POS1-set})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$(\text{false} \wedge \text{true})$
$\vee\ (\text{false} \wedge \text{true} \wedge \text{false})$
$\vee\ (\text{false} \wedge \text{true} \wedge \text{true})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : \text{false}\}$
$= \sigma_H(R)$ ♦

H-set ⊆ POS2-set

$\sigma[P](\sigma_H(R))$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : w <_P v\}$ //Def. POS/POS

$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$\quad (w[A] \in \text{POS2-set} \wedge v[A] \in \text{POS1-set})$
$\quad \vee\ (w[A] \notin \text{POS1-set} \wedge w[A] \notin \text{POS2-set} \wedge v[A] \in \text{POS2-set})$
$\quad \vee\ (w[A] \notin \text{POS1-set} \wedge w[A] \notin \text{POS2-set} \wedge v[A] \in \text{POS1-set})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$\quad (\text{true} \wedge \text{false})$
$\quad \vee\ (\text{false} \wedge \text{false} \wedge \text{true})$
$\quad \vee\ (\text{true} \wedge \text{false} \wedge \text{false})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : \text{false}\}$
$= \sigma_H(R)$ ♦

H-set ∩ (POS1-Set ∪ POS2-Set) = Ø

$\sigma[P](\sigma_H(R))$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : w <_P v\}$ //Def. POS/POS
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$\quad (w[A] \in \text{POS2-set} \wedge v[A] \in \text{POS1-set})$
$\quad \vee\ (w[A] \notin \text{POS1-set} \wedge w[A] \notin \text{POS2-set} \wedge v[A] \in \text{POS2-set})$
$\quad \vee\ (w[A] \notin \text{POS1-set} \wedge w[A] \notin \text{POS2-set} \wedge v[A] \in \text{POS1-set})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) :$
$\quad (\text{false} \wedge \text{false})$
$\quad \vee\ (\text{true} \wedge \text{true} \wedge \text{false})$
$\quad \vee\ (\text{true} \wedge \text{true} \wedge \text{false})$
$= \{w \in \sigma_H(R) \mid \neg\exists v \in \sigma_H(R) : \text{false}\}$
$= \sigma_H(R)$ ♦

Appendix 2: Operator trees of query Q_3, Q_4, Q_5 and Q_6

Example 5: *Query Q_3*

```
SELECT s.name, u.price
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref
       and c.brand = 'BMW'
PREFERRING  u.age minimal prior to
           (u.price minimal AND
            c.horsepower maximal)
```

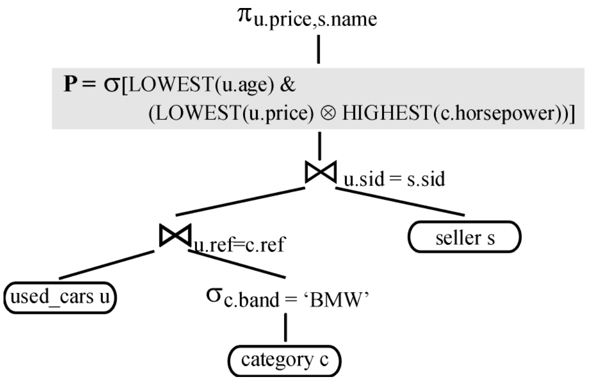


Figure 10: T_{rel} after pass 1

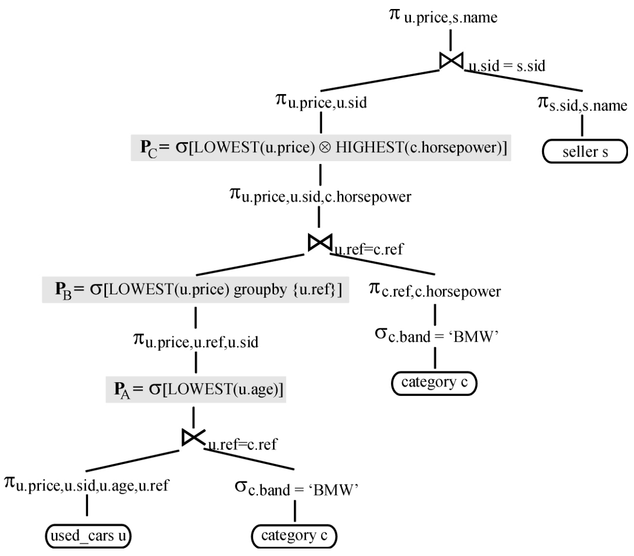


Figure 11: T_{end} after pass 2

Example 6: *Query Q_4*

```
SELECT s.name, u.price
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref
       and c.brand = 'BMW'
PREFERRING u.age minimal prior to
           u.price minimal
```

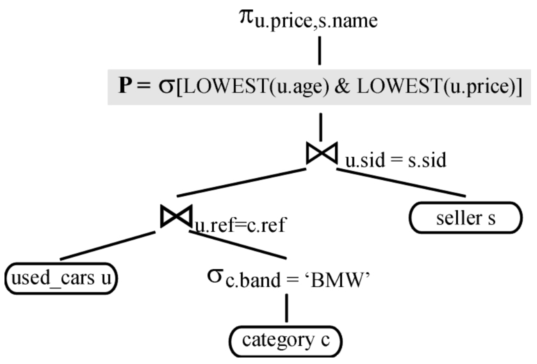


Figure 12: T_{rel} after pass 1

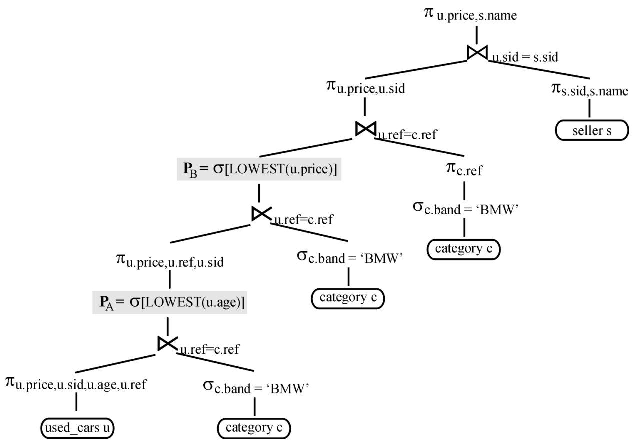
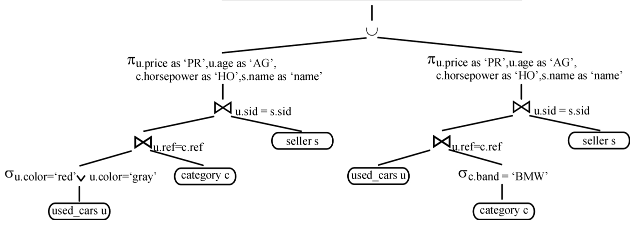


Figure 13: T_{end} after pass 2

Example 6: *Query Q_5*

```
SELECT s.name, u.price as 'pr',
       u.age as 'ag', c.horsepower AS 'hp'
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref AND
      (u.color = 'red' OR u.color = 'gray')
UNION
SELECT s.name, u.price as 'pr',
       u.age as 'ag', c.horsepower AS 'hp'
FROM   used_cars u, category c, seller s
WHERE  u.sid = s.sid AND u.ref = c.ref AND
       c.brand = 'BMW'
PREFERRING (LOWEST ag AND LOWEST pr)
           PRIOR TO HIGHEST hp;
```



P = σ[(LOWEST(AG) ⊗ LOWEST(PR))& HIGHEST(HO)]
∪
π u.price as 'PR',u.age as 'AG', c.horsepower as 'HO',s.name as 'name'
π u.price as 'PR',u.age as 'AG', c.horsepower as 'HO',s.name as 'name'
⋈ u.sid = s.sid
⋈ u.sid = s.sid
⋈ u.ref=c.ref
seller s
⋈ u.ref=c.ref
seller s
σ u.color='red' ∨ u.color='gray'
category c
used_cars u
σ c.band = 'BMW'
used_cars u
category c

Figure 14: T_{rel} after pass 1

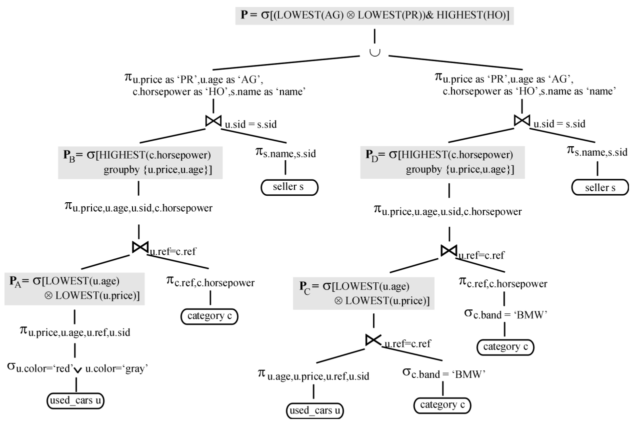

P = σ[(LOWEST(AG) ⊗ LOWEST(PR))& HIGHEST(HO)]
∪
πu.price as 'PR',u.age as 'AG', c.horsepower as 'HO',s.name as 'name'
⋈ u.sid = s.sid
PB= σ[HIGHEST(c.horsepower) groupby {u.price,u.age}]
πs.name,s.sid
seller s
πu.price,u.age,u.sid,c.horsepower
⋈ u.ref=c.ref
PA= σ[LOWEST(u.age) ⊗ LOWEST(u.price)]
πc.ref,c.horsepower
category c
πu.price,u.age,u.ref,u.sid
σu.color='red' ∨ u.color='gray'
used_cars u
πu.price as 'PR',u.age as 'AG', c.horsepower as 'HO',s.name as 'name'
⋈ u.sid = s.sid
PD= σ[HIGHEST(c.horsepower) groupby {u.price,u.age}]
πs.name,s.sid
seller s
πu.price,u.age,u.sid,c.horsepower
⋈ u.ref=c.ref
PC= σ[LOWEST(u.age) ⊗ LOWEST(u.price)]
πc.ref,c.horsepower
σc.band = 'BMW'
category c
⋉ u.ref=c.ref
π u.age,u.price,u.ref,u.sid
σc.band = 'BMW'
used_cars u
category c

Figure 15: T_{end} after pass 2

Example 7: *Query Q_6*

```
SELECT u.price FROM used_cars u, seller s
WHERE  u.sid = s.sid
PREFERRING u.price BETWEEN 5000, 30000 AND
           u.age AROUND 25 AND
           s.zipcode BETWEEN 20000, 50000;
```

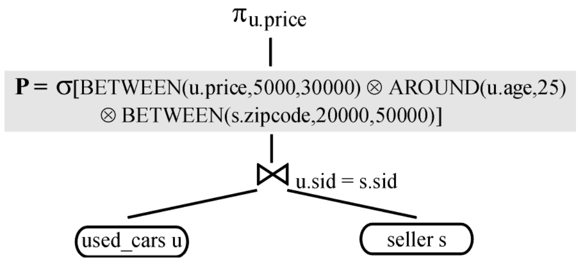

πu.price
P = σ[BETWEEN(u.price,5000,30000) ⊗ AROUND(u.age,25) ⊗ BETWEEN(s.zipcode,20000,50000)]
⋈ u.sid = s.sid
used_cars u
seller s

Figure 16: T_{rel} after pass 1

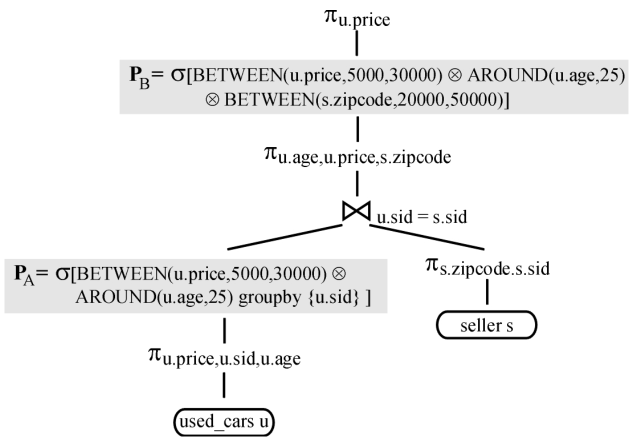

πu.price
PB= σ[BETWEEN(u.price,5000,30000) ⊗ AROUND(u.age,25) ⊗ BETWEEN(s.zipcode,20000,50000)]
πu.age,u.price,s.zipcode
⋈ u.sid = s.sid
PA= σ[BETWEEN(u.price,5000,30000) ⊗ AROUND(u.age,25) groupby {u.sid}]
πs.zipcode.s.sid
seller s
πu.price,u.sid,u.age
used_cars u

Figure 17: T_{end} after pass 2