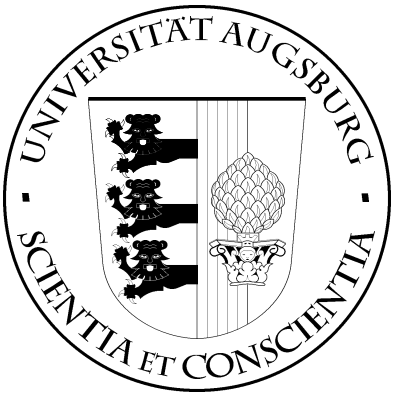
Preference SQL - Design, Implementation, Experiences

Werner Kießling, Gerhard Köstler

Angaben zur Veröffentlichung / Publication details:

Kießling, Werner, and Gerhard Köstler. 2001. “Preference SQL - Design, Implementation, Experiences.” Augsburg: Universität Augsburg.

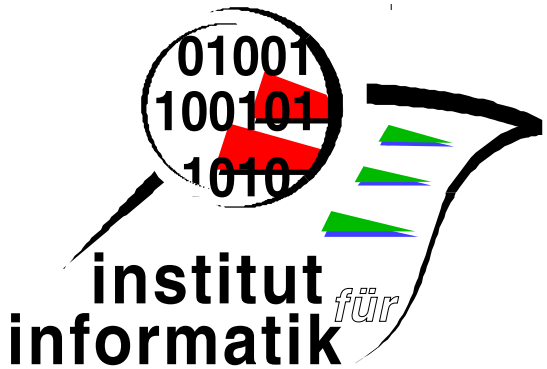
Universität Augsburg



Preference SQL — Design, Implementation, Experiences

Werner Kießling, Gerhard Köstler

Report 2001-7 Oktober 2001



Institut für Informatik
D-86135 Augsburg

Preference SQL –

Design, Implementation, Experiences

Werner Kießling[1] and Gerhard Köstler[2]

Abstract

Current search engines can hardly cope adequately with complex preferences. The biggest problem of search engines directly implemented with standard SQL is that SQL does not directly understand the notion of preferences. Preference SQL extends standard SQL by a preference model based on strict partial orders, where preference queries behave like soft selection constraints. A variety of built-in base preference types and the powerful Pareto accumulation operator to construct complex preferences, combined with the adherence to declarative SQL programming style, guarantees great programming productivity. The current Preference SQL optimizer does an efficient re-writing into standard SQL, including a high-level implementation of the skyline operator for Pareto-optimal sets. This pre-processor approach enables a seamless application integration, making Preference SQL available on a broad variety of SQL platforms including IBM DB2, Oracle, Microsoft SQL Server and Sybase. The benefits of Preference SQL technology comprise cooperative query answering and smart customer advice, leading to a higher e-customer satisfaction and shorter development times of personalized search engines for the e-service provider. We report experiences with practical applications ranging from m-commerce and comparison shopping to a large-scale performance test with real data. Several search engines of commercial B2C portals are powered by Preference SQL.

1 Introduction

When searching for items to be purchased over the Internet, **customer wishes** and **preferences** are becoming increasingly important. Just like in real shopping, a customer has his or her personal criteria and tastes that guide the search for the ideal product. These criteria can be classified into two categories: Knock out criteria that **must** be fulfilled versus **soft criteria** that **should** be fulfilled as closely as possible. Going to a real shop the customer expects to encounter a **cooperative** sales person, who assists in finding the most suitable item compatible with the stated hard and soft criteria. Positively, the same expectation for **good customer advice** carries over to e-shops in the Internet. However, the state of the art is far away from this ideal situation. Current B2C or B2B e-shops cannot cope adequately with real user preferences. As a consequence, e-shopping sessions frequently leave frustrated users behind. All too often no or no reasonable answer is returned though one has tried hard filling out query forms to match one's personal preferences closely. Most probably, one has encountered answers before sounding like "no hotels, vehicles, flights, etc. could be found that matched your criteria; please try again with different choices". The case of repeatedly receiving **empty query results** turns out to be extremely disappointing to the user, and it is even more harmful for the e-merchant. Studies by leading marketing research companies like Forrester have revealed that it requires only very few unsuccessful attempts that the user will quit, and he or she will not login to this e-shop again for quite some time.

Such an unpleasant system behavior has been recognized by e-merchants too, leading to some more or less ad hoc solutions. The simplest approach is to dictate the user to radically deviate from his initial

[1] Institute of Computer Science, University of Augsburg, kiessling@informatik.uni-augsburg.de
[2] Former CEO of Database Preference Software GmbH, Augsburg. Currently with INTERSHOP, INTERSHOP Tower, Jena, g.koestler@intershop.com

preferences by leaving some entries in the query form unspecified. In many cases instead of an empty answer the user is then **overloaded** with lots of mostly **irrelevant information**. In comparison to a real department store, this amounts to tell the customer that the desired item can be found, if at all, on maybe level 1, 3, 6 and 7 of the entire sales complex. Another approach to remedy this unpleasant situation is called parametric search. Here the user is interactively guided through the search process to narrow down the search space. At each step of this iterative procedure the system reports back whether the current result set has already become empty. If so, the user has to backtrack and retry the search along a different search path. Though reducing the frequency of empty query result, the query process can still be very time-consuming and constantly requires the user's attention.

Scientifically there have been various approaches to cope with these deficiencies, notably in the context of **cooperative database systems** [Mot88, CYC96, Min98]. There the technique of query relaxation has been studied in order to deal with the empty result problem. Other approaches aiming to provide soft constraints are based on fuzzy logic or case-based reasoning. None of these attempts does support a declarative and semantically intuitive model of preferences. Likewise, so far no smooth and efficient integration of preferences with SQL technology was available. Recently, [BKS01] can be considered as a move into this direction.

The rest of this paper is organized as follows. In section 2 we present the language design of the commercial product Preference SQL [Dat00], its key feature being the notion of preferences as strict partial orders. Preference SQL is an extension of standard SQL supporting a bundle of built-in base preference types and further operators, in particular Pareto accumulation, to build more complex preference types. Section 3 presents some implementation details of a plug-and-go application integration, including a large-scale performance test with real data. In section 4 we report various experiences with building personalized search engines. A summary and outlook is given in section 5.

2 Preference SQL Language Design

Search engines directly implemented with standard SQL suffer from the fact that SQL does not directly understand the notion of preferences, hence is incapable of directly supporting soft constraints. Thus any preference must be translated somehow into a hard criterion in the `WHERE` clause. In fact, the user is forced to think like an SQL database. What is really needed is an extension of standard SQL adding preferences as first class citizen language constructs. The crux of such an attempt is the choice of a proper model of preferences. Such a model must both be intuitive to the user and must have a suitable formal semantics that allows a smooth and efficient integration into the SQL world. Below we will introduce the basics of our preference model based on strict partial orders. The idea of declarative database queries with partial orders traces back to [KiG94]. The theoretical backbone that allows a smooth amalgamation with declarative database technology, guaranteeing the existence of a model theory and an equivalent fixpoint theory, has been laid in [KKT95].

2.1 A Model for Preferences

The phenomenon of preferences has at least two very different facets, where both may be present in a given real-life situation:

(1) Choices in an 'exact world':
Here all user wishes can be satisfied completely or not at all. Often the user's options are restricted to a pre-defined set of fixed choices. Typical examples are software configurations according to user profiles. In technical terms, queries in this context are **exact-match** queries with hard selection criteria, delivering exactly the dream object if it is there and otherwise reject the user's request.

(2) Choices in the 'real world':
Such choices behave quite differently. They are guided by personal preferences in the sense of wishes: Wishes are free, but there is no guarantee that they can all be satisfied at all times. In case of failure for the perfect match people are not always, but usually 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 match between one's wishes and the reality. In other words, preferences here will lead to the notion of **soft constraints**.

Therefore, successfully building personalized search engines requires the following:
(1) Hard constraints in an exact-match world: This is the strong point of standard SQL.
(2) Preference-driven choices in a real world: For this paradigm nothing comparable to the exact world technologies exists today.

We aim to fill the latter gap with Preference SQL. At the heart of this is a suitable model of preferences. Preferences in the real world show up in quite different ways as everybody is aware of. However, a careful examination of its vary nature reveals that they share a fundamental common principle. Let's examine our daily life with its abundance of preferences that may come from subjective feelings or other intuitive influences. In this familiar setting it turns out that people express their wishes frequently in terms of "**I like A better than B**". In this way people express a non-numeric ranking between A and B. This kind of preference modeling is universally applied and intuitively understood by everybody. In fact, every child learns to apply it from its earliest youth on. People are intuitively used to deal with such preferences, in particular with those that are not expressed in terms of numerical scores and calculations.
Thinking of preferences in terms of 'better-than' has a very natural counterpart in mathematics: One can map such real life preferences straightforwardly onto **strict partial orders**. Mathematically a preference P = (A, <P) is an irreflexive, transitive and asymmetric binary relation <P on the domain of values associated with an attribute set A.

2.2 Language Overview of Preference SQL

People are also used to express their wishes in a completely declarative manner. They simply don't want to be involved in technical details how their wishes are satisfied. What really matters is that wishes get fulfilled as good as possible. Preference SQL delivers this convenience to the user. Now we introduce the key features of the Preference SQL query language [Dat00]. Preference SQL pushes declarative SQL computing one decisive step forward. It compatibly extends standard SQL by introducing new language constructs that treat preferences as first class citizens.

Preference SQL = Standard SQL + Preferences (as strict partial orders)

Preferences can be constructed on the fly when issuing a query, or they can be defined as persistent objects using a Preference Definition Language. Preferences are syntactically expressed inside an SQL query block following the new keyword `PREFERRING`.

2.2.1 Built-in Preference Types

There are quite different selection criteria that turn out as preferences in the sense of strict partial orders. For this purpose Preference SQL provides several built-in base preference types, which are particularly useful for building search engines for e- or m-commerce. Let's give a survey by examples.

- **Approximation: AROUND, BETWEEN**

'AROUND' preferences favor values close to a numerical target value. This is useful when hitting the target value is not a must or hardly possible. This query returns trips taking 14 days if possible, else those with the closest duration to 14:

```
SELECT * FROM trips PREFERRING duration AROUND 14;
```

The 'BETWEEN[low, up]' preference type behaves analogously: Values inside [low, up] are best, otherwise being closer to the interval limits is considered better.

- **Minimization/Maximization: LOWEST, HIGHEST**

A frequently occurring preference is asking for highest or lowest values, if possible. Otherwise the closest value to the maximum or minimum, resp., is considered acceptable. The following preference query asks for the largest apartment available:

```
SELECT * FROM apartments PREFERRING HIGHEST(area);
```

This query has a simple standard SQL counterpart. But note that instead of a single attribute (like area) an *arithmetic expression* over several attributes or even a proper *stored procedure* are admissible, too.

- **Favorites, dislikes: POS, NEG**

A POS preference expresses a soft condition that a desired value *should* be one out of a given list of values. This query looks for a programmer who should have Java or C++ experience. If such an applicant does not exist, programmers with other skills will be considered alternatively.

```
SELECT * FROM programmers PREFERRING exp IN ('java', 'C++');
```

"*Should not* have" criteria are supported by NEG preferences. This query expresses a preference for a hotel outside downtown. If only hotels in downtown have rooms left, offering one of those is better than offering nothing.

```
SELECT * FROM hotels PREFERRING location <> 'downtown';
```

In the current release Preference SQL 1.3 various built-in combinations of POS and NEG preferences (e.g. POS/POS, POS/NEG) and a base preference type CONTAINS on text attributes for simple full-text search (cmp. also [LeK99]) are supported, too. Any preference that can be expressed by a finite set of 'A is better than B' relationships can be created as a base preference of type EXPLICIT.

2.2.2 Assembling Complex Preferences

In general decisions are not based on a single preference, but on a possibly complex combination of preferences. Preference SQL offers means to inductively assemble complex preferences.

- **Equal importance: Pareto accumulation (AND)**

Pareto accumulation of preferences P_1, …, P_n into a complex preference P is defined as:

$v = (v_1, \ldots, v_n)$ is better than $w = (w_1, \ldots, w_n)$ iff
$\exists$ i such that v_i is better than w_i and v is equal or better than w in any other component value

The intuitive semantics of Pareto accumulation is a non-discriminating combination of **equally important preferences** P_1, …, P_n. Pareto accumulation forms a strict partial order again, hence is a preference in our sense. The maximal values of P are called **Pareto-optimal set**. The Pareto-optimality principle has been applied and studied extensively for at least 50 years for multi-attribute decision problems in the social and economic sciences. Preference SQL's syntax for Pareto accumulation is the 'AND'-ing of base preferences.

Imagine, when buying a computer a customer considers a maximum memory size and CPU speed as equally important. This preference can be expressed as:

```
SELECT * FROM computers
PREFERRING HIGHEST(main_memory) AND HIGHEST(cpu_speed);
```

- **Ordered importance: Cascading of preferences (CASCADE)**

Cascading of preferences assigns different levels of importance to the constituent preferences, applying preferences one after the other. Preference SQL's syntax for ordered importance is 'CASCADE' (or as a synonym ',').

Suppose someone wants to buy a computer, where its color should be black or brown, which in turn is less important than a maximal size of the main memory. This wish can be expressed as:

```
SELECT * FROM computers
PREFERRING HIGHEST(main_memory) CASCADE color IN ('black','brown');
```

- **Combining Pareto accumulation and cascading**

The full power of Preference SQL comes when combining its basic constructs into complex wishes. To give an impression of its intuitive expressiveness let's phrase a customer wish in natural language:

> "My favorite car *must* be an Opel. It *should* be a roadster, but if there is none, please no passenger car. Equally important I want to spend around DM 40,000 and the car should be as powerful as possible. Less important I like a red one. If there remain several choices, let better mileage decide."

The Preference SQL query features a hard and a soft condition, the latter combining a POS/NEG preference on category, an AROUND preference on price, a POS preference on color and a LOWEST preference on mileage. It is almost a one-to-one translation of the verbal formulation above:

```
SELECT * FROM car WHERE make = 'Opel'
PREFERRING (category = 'roadster' ELSE category <> 'passenger' AND
            price AROUND 40000 AND HIGHEST(power))
            CASCADE color = 'red' CASCADE LOWEST(mileage);
```

2.2.3 Answer Explanation

When a tuple is selected or rejected by a `WHERE` condition in standard SQL, the reason is immediately evident from the tuple's attributes: If they meet the condition it belongs to the result, otherwise not. The presence of a tuple in the result set of a Preference SQL query does not only depend upon the quality of the tuple itself, but also of its competitors. This raises the need to justify the results of a preference query, just like in real life.

For this purpose Preference SQL supports special **quality functions**, reporting which soft criteria are met by a result tuple to which extent:

- `TOP(A)` is a Boolean predicate reporting whether attribute value A is a perfect match or not.
- `LEVEL(A)` reports how far the A-value of a tuple is apart from the maximal A-value (being at level 1).
- `DISTANCE(A)` reports how far the numerical A-value of a tuple is apart from the maximal A-value (being at distance 0).

Consider this oldtimer car database and the subsequent Preference SQL query, Pareto-combining a POS/POS preference with an AROUND preference:

oldtimer(ident,	color,	age)
Maggie	*white*	*19*
Bart	*green*	*19*
Homer	*yellow*	*35*
Selma	*red*	*40*
Smithers	*red*	*43*
Skinner	*yellow*	*51*

```
SELECT ident, color, age, LEVEL(color), DISTANCE(age)
FROM   oldtimer
PREFERRING color = 'white' else color = 'yellow' AND age AROUND 40;
```

The adorned Pareto-optimal result is as follows:

■	*Selma*	*red*	*40*	*3*	*0*
■	*Homer*	*yellow*	*35*	*2*	*5*
■	*Maggi*	*white*	*19*	*1*	*21*

Thus one can see at a glance which of criteria are met by the results and how much they differ from the optimum. This marks an important improvement over the unsatisfactory behavior of many search engines that rank results by numerical scores without giving any hint what these scores mean.

2.2.4 Quality Control

If perfect matches for preferences are not available, the match-making process looks for best-possible alternative answers. Preference SQL's quality functions are also useful, if the user wants to enforce certain minimal quality standards a result tuple must satisfy. E.g., assume an e-customer is looking for a travel that starts around the 3rd of July and takes around two weeks. But he or she is not willing to accept variations above two days for each criterion. Such quality restrictions can be expressed using the 'BUT ONLY' clause of Preference SQL:

```
SELECT * FROM trips
PREFERRING start_day AROUND '1999/7/3' AND duration AROUND 14
BUT ONLY   DISTANCE(start_day)<=2 AND DISTANCE(duration)<=2;
```

Clearly, an empty result may be possible now, but this correlates with the user's explicit intension!

2.2.5 The Preference SQL Query Block

The complete syntax of the current version Preference SQL 1.3 is given in [Dat00]. In general, the Preference SQL query block offers the following options, allowing for hard and soft selections to co-exist within one single query:

```
SELECT     <selection>
FROM       <table_references>
WHERE      <where-conditions>
PREFERRING <soft-conditions>
GROUPING   <attribute_list>
BUT ONLY   <but_only_condition>
ORDER BY   <attribute_list>
```

The elements that extend standard SQL appear in bold (<selection> is bold, because quality functions can appear there). Like in standard SQL, the `WHERE` and `ORDER BY` clauses are optional. Without a `PREFERRING` clause, it is not a preference query. The `GROUPING` (performing with soft constraints what `GROUP BY` does with hard constraints) and `BUT ONLY` clauses are both optional. Preferences only apply to tuples fulfilling the `WHERE` condition. The condition of the `BUT ONLY` clause is logically tested after applying the preferences of the `PREFERRING` clause. Preference SQL queries can also be invoked as sub-queries of `INSERT` statements. As a current restriction sub-queries in the `WHERE` clause may not contain `PREFERRING` clauses.

The answer semantics of Preference SQL follows a **'Best Matches Only' (BMO)** query model:

- Find all perfect matches wrt. preference P in the `PREFERRING` clause. If this set is non-empty, we are done.
- Otherwise, consider all other values within the BUT ONLY quality threshold, but discard worse values wrt. <P on the fly. All non-dominated values are returned.

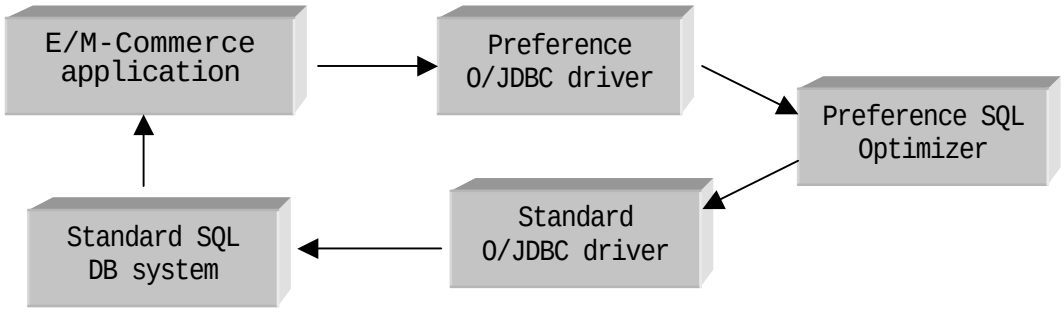
Note that in case of Pareto accumulation BMO returns exactly the Pareto-optimal set. The SQL extension proposed in [BKS01], using a SKYLINE clause, is a subset of Preference SQL.

3 Implementation of Preference SQL

Preference SQL has been designed and implemented starting late 1997 ([KiK97]) with its first commercial product release available in the fall of 1999.

3.1 Plug-and-Go Application Integration

Preference SQL is implemented as an intermediate layer between the application and the database. It processes preference queries by translating them to standard SQL queries and submitting them to the database. Queries without preferences are just passed through to the database system without causing any noticeable overhead. Legacy SQL applications run without any restriction.



In the current implementation of Preference SQL 1.3 a Preference ODBC or JDBC driver is placed directly in front of the Preference SQL Optimizer translating Preference SQL (as generated through a graphical user interface) into standard SQL, *pre-optimizing* its novel functionality. The standard ODBC or JDBC driver forwards the transformed SQL program to the underlying SQL database system, where it is *optimized a second time* by the standard SQL optimizer. The executable program is run against the existing SQL database. Any additional code, generated for query rewriting by the Preference SQL Optimizer, is fully SQL92 entry-level compliant. Thus Preference SQL can run in combination with any SQL92 entry-level compliant database system**.**

3.2 The Preference SQL Optimizer

Complex preferences can be formulated declaratively within a single preference query. It becomes the burden of the Preference SQL Optimizer to find the right answers under the BMO query model. Recently various methods have been proposed to implement the *skyline operator*, which can be employed to compute Pareto-optimal sets in special cases ([BKS01], [TEO01]). For the scope of this paper we demonstrate how to efficiently compute the Pareto-optimal answer set in our re-writing approach, piggybacking on the power of the host SQL system. The abstract algorithm is as follows:

Selection method for retrieving all maximal tuples from a relation R wrt. a strict partial order P:

(1) At the start the set of maximal tuples Max is empty.
(2) Select a tuple t1 from the R.
(3) Insert t1 into Max if there is no tuple t2 in R that is better than t1 wrt. P.
(4) Repeat steps (2) through (3) until all tuples t1 from R have been selected.
(5) The method is finished. Max contains the maximal tuples from R wrt. P.

We pose this Preference SQL query against the subsequent relation Cars:

```
SELECT * FROM Cars PREFERRING Make = 'Audi' AND Diesel = 'yes';
```

Identifier	Make	Model	Price	Mileage	Airbag	Diesel
1	Audi	A6	40000	15000	yes	no
2	BMW	5 series	35000	30000	yes	yes
3	Volkswagen	Beetle	20000	10000	yes	no

Our abstract selection method can be implemented by the following SQL92-compliant query, after proper creation of a temporary relation Max:

```
CREATE VIEW Aux AS
SELECT *, CASE WHEN Make = 'Audi' THEN 1 ELSE 2 END AS Makelevel,
          CASE WHEN Diesel = 'yes' THEN 1 ELSE 2 END AS Diesellevel
FROM Cars;

INSERT INTO Max
SELECT Identifier, Make, Model, Price, Mileage, Airbag, Diesel
FROM Aux A1
WHERE NOT EXISTS (SELECT 1 FROM Aux A2
                  WHERE A2.Makelevel <= A1.Makelevel AND
                        A2.Diesellevel <= A1.Diesellevel AND
                       (A2.Makelevel < A1.Makelevel OR
                        A2.Diesellevel < A1.Diesellevel));
```

Having the right indices available current SQL optimizers can efficiently process this SQL query with a correlated NOT EXISTS sub-query as we will demonstrate right now.

3.3 A Large Scale Performance Benchmark

One of the busiest Internet sites in Germany is a job search engine, where millions of professional skill profiles of unemployed people are online accessible. We have benchmarked Preference SQL against this highly complex system.

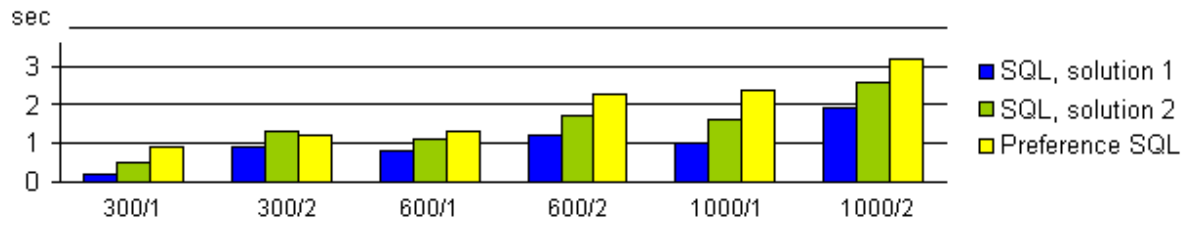
Benchmark description:

- Hardware: IBM Risc-Workstation 43P-140 running AIX 4.2 **(**mono-processor CPU, comparable to a 332 Mhz Pentium II), 768 MBytes main memory, 1 ultra SCSI hard disk.
- SQL database**:** Informix Universal Server 9.1, 1 relation with nearly 1.4 million tuples, each having 74 attributes describing the profile of a job applicant.

The search scenario of the search engine is as follows: In a pre-selection a set of hard criteria has to be filled into the search mask. If the result set size is below a default number of hits, a second selection with more job details can be issued to narrow down the candidate set. Thus benchmark queries were designed as follows (due to space limitations we cannot list the complex queries): The pre-selection is turned into hard conditions in the WHERE –clause in any case, whereas the second selection is treated differently:

- SQL, solution 1: Translation into 4 *conjunctive* conditions in the WHERE-clause.
- SQL, solution 2: Translation into 4 *disjunctive* conditions in the WHERE-clause.
- Preference SQL: Translation into 4 *Pareto-accumulated* conditions in the PREFERRING-clause.

Here are the real time measurements for results set sizes of 300, 600 and 1000 for the pre-selection and for two different conditions chosen for the second selection:



We think that these performance results impressively demonstrate the efficiency of our high-level approach for implementing Pareto-optimality. Further optimizations not mentioned here or implementing a generalized skyline operator in the kernel of an SQL-system clearly hold much promise for additional speed-ups.

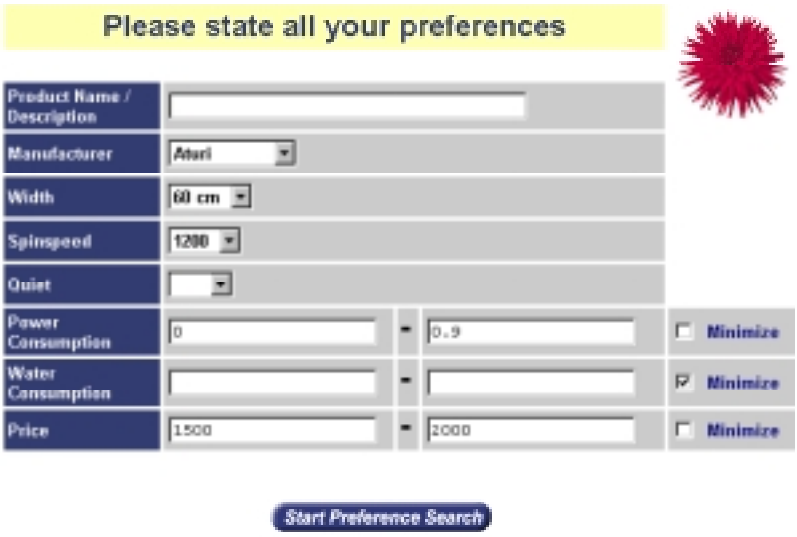
4 Experiences with Preference SQL

Preference SQL is useful for all database applications where customer wishes are naturally modeled by preferences as opposed to hard conditions only. This applies for most e-market places and products that have complex properties and limited availability, like e.g. used cars, flights, hotels, computers, real estate or jobs. Let us demonstrate the power of Preference SQL from different angles.

4.1 Building E-Shopping Search Engines

When designing a personalized search engine the issue of **preference modeling** becomes crucial:

- Which selection criteria are hard (WHERE clause) vs. which are soft (PREFERRING clause)?
- Which quality control (BUT ONLY condition)?
- Importance of criteria (Pareto accumulation vs. cascading)?
- Where do preferences come from: Hard-wired into search mask as determined by the e-tailer vs. determined by the e-customers etc.?



Given this sample search mask for an e-shop selling washing machines, let's assume that all preference modeling decisions are invisibly hard-wired into the design of the search mask.
Using dynamic Preference SQL it is straightforward to generate the subsequent Preference SQL query from a given user input. Note that an e-merchant has complete freedom to add further so-called *vendor preferences*, maybe on hidden attributes, to this query at his discretion.

```
SELECT * FROM products WHERE manufacturer = 'Aturi'
PREFERRING (width AROUND 60 AND spinspeed AROUND 1200) CASCADE
     (powerconsumption BETWEEN 0, 0.9 AND LOWEST(waterconsumption)
      AND price BETWEEN 1500, 2000;
```

If the query result should, e.g., be displayed with highlighted perfect attribute matches, the query can be enhanced with quality functions (not shown above).

Search engine technology based on Preference SQL has been integrated as **Preference Search** cartridge into the leading e-commerce platform INTERSHOP 4 (running on Sybase ASE) and INTERSHOP enfinity (running on Oracle 8i). In this way it has been deployed in the e-commerce market, e.g. a very successful German e-portal for office-supply and the regional marketplace www.msp-info.de are powered by Preference SQL.

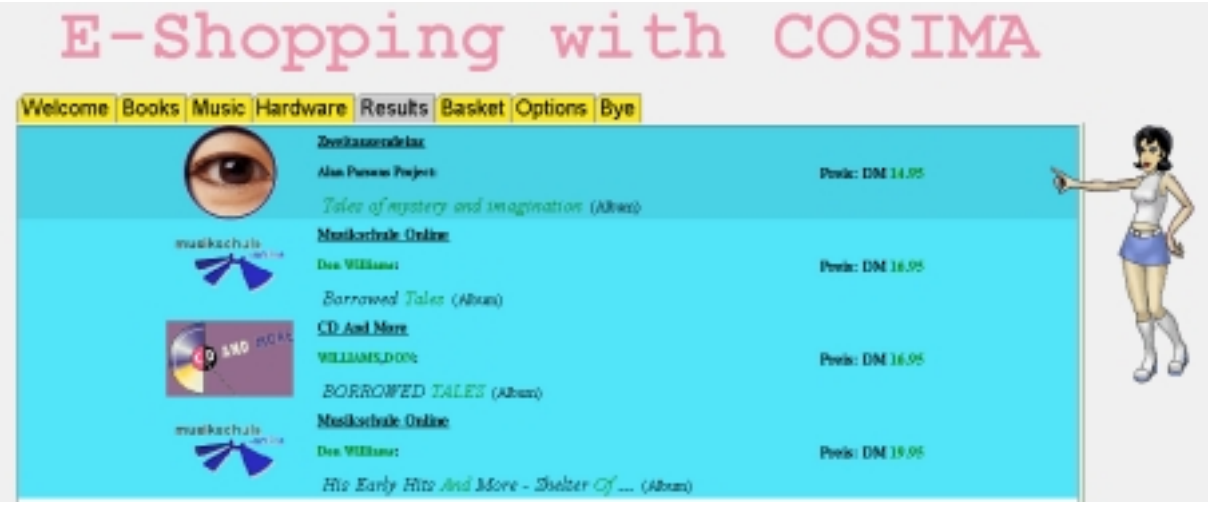
4.2 Mobile Search

In mobile environments including PDAs, WAP or I-mode phones the quality of a search engines matters even more. Retrying queries by tiresome typing because of empty results or lengthy scrolling through lots of unwanted results are major obstacles towards successful m-commerce. Obviously, a search engine based on Preference SQL's BMO query model is an appealing choice, in particular in combination with location-based e-services: The first query delivers already the best possible results only. This does not only save typing effort, but also mobile phone costs!

4.3 Advanced Cooperative Sales Interfaces

The answer explanation capabilities of Preference SQL lay the foundation for advanced **cooperative e-sales interfaces** as recently demonstrated by the non-commercial e-commerce application COSIMA ([KFH01]). COSIMA (downloadable free of charge from www.myCOSIMA.com) features a meta-search engine for **comparison shopping** over various well-known e-shops like Amazon, BOL, etc. Intermediate query results, gathered by agent technology over the Internet from the participating e-shops, are stored in a temporary COSIMA database running Preference SQL on Oracle 8i. The discussed salient characteristics of Preference SQL enable a novel type of cooperative sales interface:

In the spirit of shops of the old economy the charming avatar COSIMA, talking by dynamically generated smart **speech output** with the e-customer, performs the presentation of query results. Doing so, COSIMA explains the quality of presented items, augmented by ingredients of sales psychology.



COSIMA has been exposed already to a large general audience at the computer fair SYSTEMS 2000 in Munich and at the recent SIGMOD conference ([KHF01]). Feedback gathered so far and a growing community of many hundreds of user strongly indicate that this is a promising path to pursue for next generation e/m-commerce systems. Technically, the results in [KFH01] give strong evidence that Pareto accumulation is an indispensable operator when dealing with soft constraints. E.g., predominantly the size of the Pareto-optimal set was between 1 and 20, yielding an easy-to-survey choice of products similar to a traditional sales situation. Performance-wise the whole meta-search with Preference SQL consumed 1-2 seconds on the average, adding only a small overhead to the total response times, dominated by accessing the participating e-shops.

5 Summary and Outlook

We have presented an overview of Preference SQL that compatibly extends standard SQL with preferences under a strict partial order semantics. Salient features include a variety of built-in base preference types, the Pareto accumulation constructor to assemble complex preferences and quality control functions. We gave some insights into the Preference SQL optimizer and presented a large-scale performance benchmark, indicating that extending SQL by soft constraints can be implemented with efficiently. In particular we gave an efficient high-level re-writing method for implementing a generalized skyline operator. By selected applications ranging from comparison shopping to m-commerce we gave strong evidence that cooperative database interfaces can substantially benefit from Preference SQL which has been fully operational as commercial product since 1999.

Preference SQL is an instance of a comprehensive research effort in progress. An even richer preference type system (including numerical ranking) together with a preference algebra are being investigated under the motto "*It's a Preference World*" ([Kie01]). This preference model is being implemented for the XML world by means of Preference XPATH ([KHF01]).

Acknowledgments:
We would like to thank P. Rieger, J. Seidel and J. Wunderwald for their valuable contributions throughout the product development of Preference SQL.

References:

[BKS01] S. Borzsonyi, D. Kossmann, K. Stocker: *The Skyline Operator*. Proc. 17th Intern. Conf. On Data Engineering, Heidelberg, Germany, April 2001.

[CYC96] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, C. Larson: *CoBase - A Scalable and Extensible Cooperative Information System.* Journal of Intelligent Information Systems, 6(3):223-259, 1996.

[Dat00] Database Preference Software GmbH: *Preference SQL User Manual 1.3*, Augsburg, Germany, 2000.

[Min98] J. Minker: *An Overview of Cooperative Answering in Databases*. Proc. 3rd Intern. Conf. on Flexible Query Answering Systems, Springer LNCS 1495, pp. 282-285, Roskilde, Denmark, 1998.

[Mot88] A. Motro: *VAGUE - A User Interface to Relational Databases that Permits Vague Queries*. ACM Transactions on Office Information Systems , 6:187-214, 1988.

[Kie01] W. Kießling: *Foundations of a Preference World (not only for Database Systems)*. Monograph in preparation, Univ. of Augsburg.

[KiG94] 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, Springer LNCS 777, pp. 118-138, Hamburg, 1994.

[KiK97] W. Kießling, G. Köstler: *Preference SQL – A New Paradigm for Online Information Systems*. Internal document, Database Preference Software GmbH, Augsburg, Oct. 1997.

[KHF01] W. Kießling, S. Holland, S. Fischer, T. Ehm: *COSIMA - Your Smart, Speaking E-Salesperson*. Proc. ACM SIGMOD Intern. Conf. on Management of Data, demo paper, p. 600, Santa Barbara, USA, May 2001.

[KFH01] W. Kießling, S. Fischer, S. Holland, T. Ehm: *Design and Implementation of COSIMA - A Smart and Speaking E-Sales Assistant*. Proc. 3rd Intern. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, pp. 21-30, San Jose, USA, June 2001.

[KHF01] W. Kießling, B. Hafenrichter, S. Fischer, S. Holland: *Preference XPATH: A Query Language for E-Commerce*. Proc. 5th Intern. Konferenz für Wirtschaftsinformatik, Augsburg, Germany, pp. 425-440, Sept. 2001.

[KKT95] G. Köstler, W. Kießling, H. Thöne, U. Güntzer: *Fixpoint Iteration with Subsumption in Deductive Databases*. In Journal of Intelligent Information Systems, Vol. 4, pp. 123-148, Boston, USA, 1995.

[LeK99] A. Leubner, W. Kießling: *Personalized Nonlinear Ranking Using Full-text Preferences*. Proc ACM SIGIR Workshop on Customised Information Delivery, Berkeley, USA, 1999.

[TEO01] K.-L. Tan, P.-K. Eng, B. C. Ooi: *Efficient Progressive Skyline Computation*. Proc. 27th Intern. Conf. on Very Large Databases, pp. 301-310, Rome, Italy, Sept. 2001.