Personalized Search And Recomendations For Movies In A Relational Database

Harish.T[1],Jaya Surya.K[2],Mr.P.Balamurugan[3]

Student, Dept. of Computer Science and Engineering, Agni College of Technology, India.[1,2]
Asst. Professor, Dept. of Computer Science and Engineering, Agni College of Technology, India. [3]

ABSTRACT - *We argue that preference-aware query processing needs to be pushed closer to the DBMS. We introduce a preference-aware relational data model that extends database tuples with preferences and an extended algebra that captures the essence of processing queries with references. Based on a set of algebraic properties and a cost model that we propose, we provide several query optimization strategies for extended query plans. Further, we describe a query execution algorithm that blends preference evaluation with query execution, while making effective use of the native query engine. We have implemented our framework and methods in a prototype system, PrefDB. PrefDB allows transparent and efficient evaluation of preferential queries on top of a relational DBMS. Our extensive experimental evaluation on two real-world data sets demonstrates the feasibility and advantages of our framework.*

KEYWORDS: **Query processing,Relational data model, Query optimization, Prototype system, PrefDB**

1. INTRODUCTION:

The main aim of this project is to propose and provide several query optimization strategies for extended query plans and describe a query execution algorithm that blends preference evaluation with query execution, while making effective use of the native query engine.

PrefDB(Preference database), a preference-aware relational system that transparently and efficiently handles queries with preferences. In its core, PrefDB employs a preference-aware data model and algebra, where preferences are treated as first-class citizens. We define a reference using a condition on the tuples affected, a scoring function that scores these tuples, and a confidence that shows how confident these scores are.

In our data model, tuples carry scores with confidences. Our algebra comprises the standard relational operators extended to handle scores and confidences. For example, the join operator will join two tuples and compute a new score-confidence pair by combining the scores and confidences that come with the two tuples. In addition, our algebra contains a new operator, prefer, that evaluates a preference on a relation, i.e., given as inputs a relation and a preference on this relation, prefer outputs the relation with new scores and confidences. During preference evaluation, both the conditional and the scoring part of a preference are used. The conditional part acts as 'soft' constraint that determines which tuples are scored without disqualifying any tuples from the query result.

In this way, PrefDB separates preference evaluation from tuple filtering. This separation is a distinguishing feature of our work with respect to previous works. It allows us to define the algebraic properties of the prefer operator and build generic query optimization and processing strategies that are applicable regardless of the type of reference specified in a query or the expected type of answer.

2. SCOPE OF THE PROJECT

We have developed PrefDB, a preference-aware relational system that transparently and efficiently handles queries with preferences. In its core, PrefDB employs a preference-aware data model and algebra, where preferences are treated as first-class citizens.We define a reference using a condition on the tuples affected, a scoring function that scores these tuples, and a confidence that shows how confident these scores are. In our data model, tuples carry scores with confidences. Our algebra comprises the standard relational operators extended to handle scores and confidences. For example, the join operator will join two tuples and compute a new score-confidence pair by combining the scores and confidences that come with the two tuples. In addition, our algebra contains a new operator, prefer, that evaluates a preference on a relation, i.e., given as inputs a relation and a preference on this relation, prefer outputs the relation with new scores and confidences. During preference evaluation, both the conditional and the scoring part of a preference are used. The conditional part acts as 'soft' constraint that determines which tuples are scored without disqualifying any tuples from the query result. In this way, PrefDB separates preference evaluation from tuple filtering. This separation is a distinguishing feature of our work with respect to previous works. It allows us to define the algebraic properties of the prefer operator and build generic query optimization and processing strategies that are applicable regardless of the type of reference specified in a query or the expected type of answer.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Several approaches to integrating preferences into database queries have been proposed and can be roughly divided into two categories. Plug-inapproaches operate on top of the database engine and they typically translate preferences into conventional query constructs. On the other hand, native approaches focus on supporting more efficiently specific queries, such as top-k or skyline queries, by injecting new operators inside the database engine. Unfortunately, both approaches have several limitations. In plug-in methods, the way preferences will be used, for example as additional query constraints or as ranking constructs, the query execution flow as well as the expected type of answer (e.g., top-k or skyline) are all hard-wired in the method, which hinders application development and maintenance.

On the other hand, native methods consider preference evaluation and filtering as one operation. Due to this tight coupling, these methods are also tailored to one type of query. Furthermore, they require modifications of the database core, which may not be feasible or practical in real life. Overall, both native and plug-in approaches do not offer a holistic solution to flexible processing of queries with preferences.

3.2 PROPOSED SYSTEM

We first construct an extended query plan that contains all operators that comprise a query and we optimize it. Then, for processing the optimized query plan, our general strategy is to blend query execution with preference evaluation and leverage the native query engine to process parts of the query that do not involve a prefer operator. Given a query with preferences, the goal of query optimization is to minimize the cost related with preference evaluation.

Based on the algebraic properties of prefer, we apply a set of heuristic rules aiming to minimize the number of tuples that are given as input to the prefer operators. We further provide a cost-based query optimization approach. Using the output plan of the first step as a skeleton and a cost model for preference evaluation, the query optimizer calculates the costs of alternative plans that interleave preference evaluation and query processing in different ways. Two plan enumeration methods, i.e., a dynamic programming and a greedy algorithm are proposed. For executing an optimized query plan with preferences, we describe an improved version of our processing algorithm (GBU) (an earlier version is described in. The improved algorithm uses the native query engine in a more efficient way by better grouping operators together and by reducing the out-of-the-engine query processing.

4. ALGORITHM

Our proposed system designed with RC4 or with the AES in CBC mode we use to first describe RC4 Algorithm with different steps for of our joint E/W algorithm. In the encrypted domain, bits of Msg will be extracted from the SHA-1 signature of these blocks. With the RC4 algorithm, it is possible to work with smaller pixel block dimension due to the fact that it works on stream of bytes. In the encrypted domain, the encrypted image is decomposed in blocks of N bytes. Then, the function f is applied to each block to extract one bit of Msg. In the spatial domain, the message Msg is extracted based on principles of the QIM.

5. IMPLEMENTATION

5.1 REGISTRATION & INTEREST SUM UP:

During Registration, each and every user will provide their basic information for authentication. After that, user has to provide their profile information and their interests about their movie. Based upon their, and with our movie datasets, we can be able to analyze their interest about the movie and have to provide the recommended movies to the particular user.

5.2 QUERY FORMATION:

A preferential query combines p-relations, extended relational and prefer operators and returns a set of tuples that satisfy the boolean query conditions along with their score and confidence values that have been calculated after evaluating all prefer operators on the corresponding relations. Intuitively, the better a tuple matches preferences and the more (or more confident) preferences it satisfies, the higher its final score and confidence will be, respectively.

The query parser adds a prefer operator for each preference. Finally, the query parser checks for each preference, whether it involves an attribute (either in the conditional or the scoring part) that does not appear in the query and modifies project operators, such that these attributes will be projected as well.

5.3 QUERY OPTIMIZATION & EXECUTION:

Extended relational operators and the prefer operator do not change how tuples are filtered or joined; for instance, prefer operator does not filter any tuples. Therefore our extended relational operators do not affect the non preference related cost. Thus, we can expect that the join order that is suggested by the native query optimizer for a query if no prefer operators were present, will still yield good performance for the non-preference part of the same query with the prefer operators. Based on this observation, we will keep the suggested join order and we will consider the non-preference related cost as fixed.

Then, the goal of our query optimizer will be to minimize the cost related with preference evaluation. Typically, the most critical parameter that shapes the processing cost of query evaluation is the disk I/Os, which is proportional to the number of tuples flowing through the operators in the query plan. Assuming a fixed position for the other operators, the goal of our query optimizer is essentially to place the prefer operators inside the plan, such that the number of tuples flowing through the score tables is minimized. The execution engine of PrefDB is responsible for processing a preferential query and supports various algorithms.

6. SYSTEM ARCHITECTURE

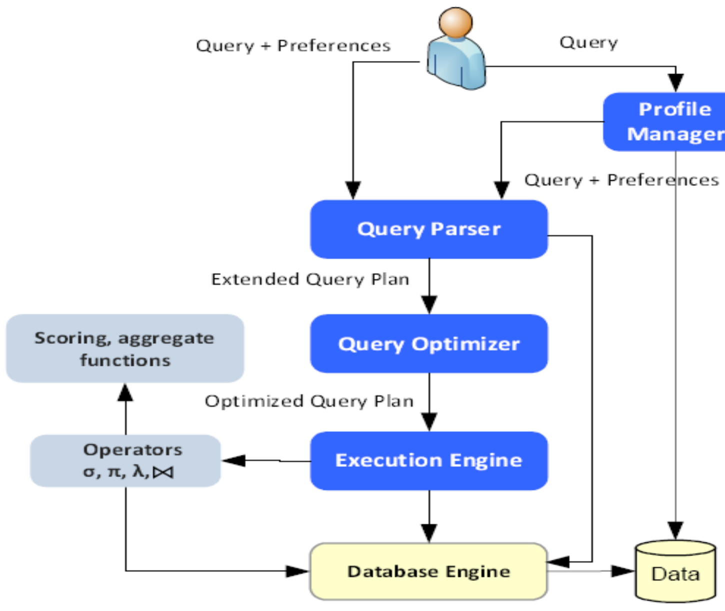

Fig 1: System Architecture

7. CONCLUSION AND FUTUREWORK

In this work we presented a preference-aware data model where preference appear as first-class citizens and preference evaluation is captured as a special 'prefer' operator. We studied the algebraic properties of the new operator and applied them in order to develop cost-based query optimizations and holistic query processing methods. Our experiments using a prototype system implementation demonstrated the performance advantages of our methods when compared with two variation of a plug-in strategy.

In Future,the enhanced preference-aware data model is presented and query optimization is done in an efficient wa with real world datasets.

REFERENCES:

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possibleextensions. TKDE, 17(6):734–749, 2005.

[2] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In SIGMOD, pages 383–394, 2006.

[3] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In SIGMOD, pages 297–306, 2000.

[4] A. Arvanitis and G. Koutrika. PrefDB: Bringing preferences closer to the DBMS. In SIGMOD, pages 665–668, 2012.

[5] A. Arvanitis and G. Koutrika. Towards preference-aware relational databases. In ICDE, pages 426–437, 2012.

[6] S. B¨orzs¨onyi, D. Kossmann, and K. Stocker. The skyline operator. In ICDE, pages 421–430, 2001.

[7] J. Chomicki. Preference formulas in relational queries. TODS, 28(4):427–466, 2003.