YmalDB: A Result-Driven Recommendation System for Databases

Marina Drosou
Computer Science Department
University of Ioannina, Greece
mdrosou@cs.uoi.gr

Evaggelia Pitoura
Computer Science Department
University of Ioannina, Greece
pitoura@cs.uoi.gr

ABSTRACT

To assist users in database exploration, we present the YmalDB system, a database system enhanced with a recommendation functionality. Along with the results of each user query, YmalDB computes and presents to the users additional results, called Ymal (i.e., "You May Also Like") results, that are highly related with the results of their original query. Such results are computed using the most interesting sets of attribute values, called *faSets*, that appear either in the results of the original query or in the results of an appropriately expanded one. The interestingness of a faSet is based on its frequency both in the query result and in the database.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation, Search process*

General Terms

Algorithms, Experimentation, Design, Performance

1. INTRODUCTION

Typically, users interact with a database system by formulating queries. This interaction mode assumes that users have a clear understanding of their information needs and the exact content of the database. However, as databases become larger and accessible to a more diverse and less technically-oriented audience, the need for an exploratory mode of interaction arises.

Some form of flexible querying is offered by solutions for handling the *many-* or *empty-* answers problems. Approaches to the many-answers problem range from reformulating the original query so as to restrict the size of its result (for example, by adding constraints to it (e.g., [9]) to automatically ranking the query results and presenting to the user only the top-k most highly ranked among them (e.g., [4]). The empty-answers problem is commonly handled by relaxing the original query (e.g., [8]). A form of exploratory search is also offered by facet queries (e.g., [7]). With facet search, users start with a general query and progressively narrow its results down to a specific item.

The "*You May Also Like*" database system (YmalDB) presented in this paper takes a different approach to exploratory database search through result recommendations [6, 10]. YmalDB computes and presents to the users additional items which, although not part of the result of their original query, may be of interest to them. This allows users to receive interesting information that they may not be aware of. For instance, when searching for movies directed by M. Scorsese, YmalDB guides exploration by recommending to users movies by other directors that have directed movies similar to those of M. Scorsese, i.e., movies with similar characteristics, such as, genre or production year. In computing related results, YmalDB considers expansions of the original query with additional attributes, by finding correlations among attributes in various relations. For example, when asking for the title of a movie, its genre or other characteristics are also considered towards discovering similar movies.

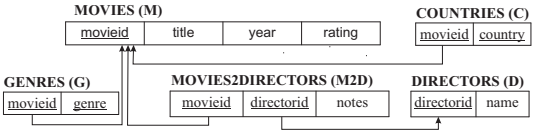
In contrast to other recent work in database recommendations (e.g., [3]), that assumes the existence of query logs, YmalDB uses only the result of the current query and the database content. In particular, the computation of recommended results is based on the most interesting sets of (attribute, value) pairs, called faSets, that appear in the result of the original user query. The *interestingness* of a faSet expresses how unexpected it is to see this faSet in the result. Interestingness depends both on the frequency of the faSet in the query result and in the database. Since determining the frequencies of all faSets in the database is computationally expensive, we maintain statistics that allow us to *estimate* those frequencies when needed. More specifically, we store information concerning only the frequencies of rare faSets, using an appropriate compact representation.

In the demonstration of YmalDB, users will be given the opportunity to submit queries (using SQL or a form-based interface) to two different databases, one with movies and one with automobiles, and receive related recommendations. Furthermore, by logging in as administrators, users will be able to tune features of the system, such as the size of the maintained statistics and potential attribute expansions.

2. THE REDRIVE FRAMEWORK

YmalDB is based on the ReDRIVE exploration framework [6]. In a nutshell, ReDRIVE database exploration works as follows. Given an SQL query Q, the top-k most

EDBT/ICDT '13 March 18 - 22 2013, Genoa, Italy



(a) Database schema.

```
SELECT M.title, M.year, G.genre
FROM D, M2D, M, G
WHERE D.name = "M. Scorsese"
      AND D.directorid = M2D.directorid
      AND M2D.movieid = M.movieid
      AND M.movieid = G.movieid
```

(b) Example query.

M.title	M.year	G.genre
The Aviator	2004	Biography
Gangs of New York	2002	Drama
Goodfellas	1990	Biography
Casino	1995	Drama
Shutter Island	2004	Thriller
M. Jackson: Video Greatest Hits	1995	Drama
The Last Waltz	1978	Biography
Raging Bull	1980	Documentary

(c) Query result.

Figure 1: Example query and result set.

interesting pieces of information in the result of Q, called *faSets*, are computed and presented to the user. These faSets are either interesting pieces (sub-tuples) of the tuples in the result of Q or extended tuples that include additional attributes not in the original result. Interesting faSets are used to construct exploratory SQL queries, whose results, called YMAL results, are recommended to the user.

2.1 Interesting FaSets

Let $\mathcal{D}$ be a relational database with n relations $\mathcal{R} = \{R_1, \ldots, R_n\}$ and $\mathcal{A}$ be the set of all attributes in $\mathcal{R}$. We use $\mathcal{A}_C$ to denote the set of categorical attributes and $\mathcal{A}_N$ to denote the set of numeric attributes, where $\mathcal{A}_C \cap \mathcal{A}_N = \emptyset$ and $\mathcal{A}_C \cup \mathcal{A}_N = \mathcal{A}$. To locate items of interest, users pose queries. In particular, we consider Select-Project-Join (SPJ) queries of the following form:

SELECT $proj(Q)$
FROM $rel(Q)$
WHERE $scond(Q)$ **AND** $jcond(Q)$

where $rel(Q)$ is a set of relations, $scond(Q)$ is a conjunction of selection predicates, $jcond(Q)$ is a set of join conditions among the relations in $rel(Q)$ and $proj(Q)$ is the set of projected attributes. A *selection predicate* is a predicate of the form $(A_i = a_i)$, where $A_i \in \mathcal{A}_C$ and $a_i \in domain(A_i)$, or of the form $(l_i \leq A_i \leq u_i)$, where $A_i \in \mathcal{A}_N$, l_i, $u_i \in domain(A_i)$ and $l_i \leq u_i$. The *result set*, $Res(Q)$, of a query Q is a relation with schema $proj(Q)$.

Let us first define pieces of information in the result set:

DEFINITION 1 (FACET AND M-FASET). *A facet condition, or simply* facet*, is a selection predicate. An m-set of facets, or m-*faSet*, $m \geq 1$, is a set of m selection predicates involving m different attributes.*

We shall also use the term faSet when the size of the m-faSet is not of interest.

For a faSet f, we use $Att(f)$ to denote the set of attributes that appear in f. Let t be a tuple from a set of tuples S with schema R; we say that t satisfies a faSet f, where $Att(f) \subseteq R$, if $t[A_i] = a_i$, for all predicates $(A_i = a_i) \in f$ and $l_i \leq t[A_i] \leq u_i$, for all predicates $(l_i \leq A_i \leq u_i) \in f$. We call the percentage of tuples in S that satisfy f, *support* of f in S. In the following, we use the term faSet to mean both the conditions and the list of the associated values appearing in the conditions.

Example: Consider the movies database, the query and its corresponding result set depicted in Fig. 1. $\{G.genre = \text{"Biography"}\}$ is a 1-faSet with support 0.375 and $\{1990 \leq M.year \leq 2009,\ G.genre = \text{"Biography"}\}$ is a 2-faSet with support 0.25.

We are looking for interesting pieces of information at the granularity of a faSet: this may be the value of a single attribute (i.e., a 1-faSet) or the values of m attributes (i.e., an m-faSet). To define faSet relevance formally, we take an IR-based approach and rank faSets in decreasing order of their odds of being relevant to a user information need.

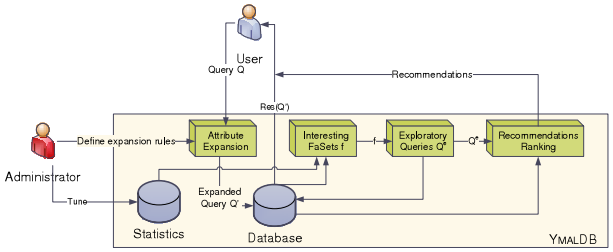


Figure 2: System architecture.

DEFINITION 2 (INTERESTINGNESS SCORE). *Let Q be a query and f be a faSet with $Att(f) \subseteq proj(Q)$. The interestingness score, $score(f, Q)$, of f for Q is defined as:*

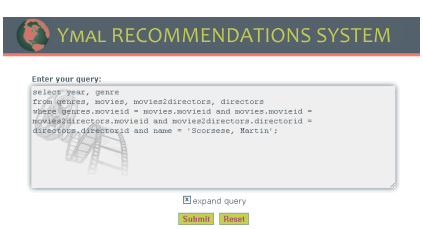
$$score(f, Q) = \frac{p(f|Res(Q))}{p(f|\mathcal{D})}$$

The term $p(f|Res(Q))$ is estimated by the support of f in $Res(Q)$, i.e., the percentage of tuples in the result set that satisfy f. The term $p(f|\mathcal{D})$ is a global measure that does not depend on the specific query. It is estimated by the support of f in $\mathcal{D}$, i.e., the percentage of all tuples that satisfy f. It serves as an indication of how frequent f is in the whole database, i.e., it measures the discriminative power of f.

Example: In the example in Fig. 1, *"Drama"* appears as frequently as *"Biography"* in the result set. However, if *"Biography"* appears less frequently than *"Drama"* in the database, then *"Biography"* is considered more interesting for this specific query than *"Drama"*.

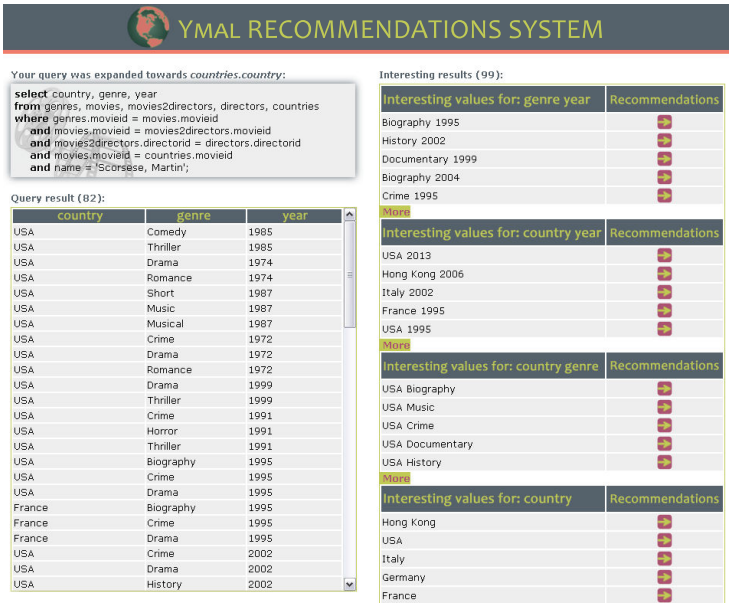
Clearly, the $scond(Q)$ part of a query is also a faSet. Another way of interpreting the interestingness score of f for Q is as the *lift* of the association rule: $scond(Q) \rightarrow f$. A high lift value indicates a strong dependency of the faSet f on the selection conditions of Q.

To estimate $p(f|\mathcal{D})$, we maintain statistics concerning rare faSets, i.e., faSets that appear in the database less frequently than a system-defined threshold ξ. Since the number of rare faSets is very large, we maintain a tunable number of ϵ-tolerance closed rare faSets [5, 6] and use them to *estimate* the frequency of any faSet in the database. Furthermore, since most real datasets have a large number of rare faSets that appear only a few times (most often only once), to avoid generating all super-sets of such faSets, we employ a hash-based data structure (in particular, a Bloom filter). In a single scan, we identify all faSets that appear less frequently than a system-defined threshold ξ_0, insert them in the Bloom filter and do not consider them any further. Then, we proceed with the generation of rare faSets using ξ.



(a) Submitting queries.

(c) YMAL Recommendations.



(b) Query results and interesting faSets.

Figure 3: YmalDB user interface.

Attribute Expansion. Definition 2 provides a means of ranking the various faSets that appear in the result set of a query Q. However, there may be interesting faSets that include attributes that are not included in $proj(Q)$ and, thus, do not appear in $Res(Q)$. For example, take a query Q that just returns the titles of movies directed by M. Scorsese. All faSets appear only once in the result set of Q. However, including for instance the relation "Countries" in $rel(Q)$ (and modifying $jcond(Q)$ accordingly) may disclose interesting information, e.g., that many of the movies directed by M. Scorsese are related to Italy.

To discover such potentially interesting faSets, we extend the definition of interestingness to include faSets with attributes not in $proj(Q)$ by introducing an extended query Q' with the same $scond(Q')$ as the original query Q but with additional attributes in $proj(Q')$ (and the corresponding additional relations in $rel(Q')$). These attributes are dictated by *expansion rules*. An expansion rule is a rule of the form $A \rightarrow B$, where A is a set of attributes in the user query, i.e., $A \subseteq proj(Q)$ and B is such that $B \subseteq \mathcal{A} \backslash proj(Q)$. The expansion rule indicates that when a query Q contains all attributes of A in its select clause, Q may be expanded to create a query Q' that also contains the attributes of B. Interesting faSets are sought in the result of Q'.

Example: Consider the query Q of Fig. 1(b) and the expansion rule $\{M.year, G.genre\} \rightarrow \{C.country\}$. We construct Q' with $proj(Q') = proj(Q) \cup \{C.country\}$ and $rel(Q') = rel(Q) \cup \{C\}$.

2.2 Exploratory Queries

We use interesting faSets to discover additional pieces of data that are potentially of interest to the user. In particular, we aim at constructing exploratory queries that retrieve results strongly correlated with those of the original user query Q by replacing the selection conditions, $scond(Q)$, of Q with related ones. Recall that a high interestingness score for f means that the lift of $scond(Q) \rightarrow f$ is high, indicating a high correlation between $scond(Q)$ and f.

For example, by replacing the $scond(Q)$ of Q in Fig. 1 with its interesting faSet $\{$*"Biography"*$\}$, we get the exploratory query:

```
SELECT D.name
FROM D, M2D, M, G
WHERE G.genre = 'Biography'
  AND D.name <> 'M. Scorsese'
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
  AND M.movieid = G.movieid
```

which retrieves additional directors that have directed biography movies, which is an interesting value appearing in the original query result set. The negation term "D.name <> M. Scorsese" is added to prevent values appearing in the selection conditions of the original user query from being recommended to the users, since these are already known to the user. An exploratory query is formally defined as follows.

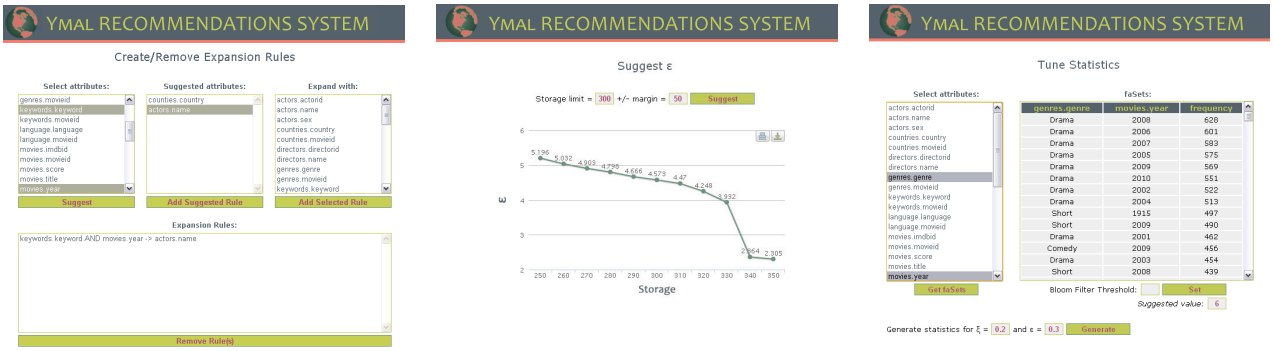
DEFINITION 3 (EXPLORATORY QUERY). *Let Q be a user query and f be an interesting faSet for Q. The exploratory query Q^e that uses f is an SPJ query with $proj(Q^e) = Attr(scond(Q))$, $rel(Q^e) = rel(Q) \cup \{R^e \mid A_i \in R^e$, for $A_i \in Att(f)\}$, $scond(Q^e) = f \wedge \neg\, scond(Q)$ and $jcond(Q^e) = jcond(Q) \wedge$ (joins with $\{R^e \mid A_i \in R^e$, for $A_i \in Att(f)\}$).*

The results of the exploratory query Q^e are recommended to the user. Clearly, one can use the interesting faSets in the results of an exploratory query to construct other exploratory queries. This way, users may start with an initial query Q and gradually discover other interesting information in the database through results attained by applying exploratory queries progressively.

3. THE YMALDB DEMONSTRATION

YMALDB is implemented in Java on top of MySQL. Our system architecture is shown in Fig. 2. YMALDB can be accessed via a simple web browser using an intuitive GUI. During the demonstration, users will be allowed to submit queries and see recommendations.

Two real datasets will be used: (i) "`MOVIES`", a database with 13 relations whose sizes range from around 10,000 to



(a) Defining expansion rules. (b) Suggesting parameter values. (c) Tuning statistics.

Figure 4: YmalDB administrator interface.

almost 1,000,000 tuples, containing information extracted from the Internet Movie Database [1] and (ii) "AUTOS", a single-relation database consisting of 12 characteristics for 15,191 used cars from Yahoo!Auto [2].

Users will be allowed to submit their queries via SQL or by employing available input forms (Fig. 3(a)). Users are first presented with the results of their query and with a list of the interesting faSets of these results (Fig. 3(b)). The interesting faSets are grouped in categories according to the attributes they contain with longer faSets presented higher in the list. The faSets in each category are ranked in decreasing order of their interestingness score and the top-5 faSets of each category are displayed. The user can click a "More" button to see more faSets.

Next to each interesting faSet, an arrow button appears. When the user clicks on it, a set of YMAL results (Fig. 3(c)), i.e., recommendations, appear in a separate window. These recommendations are retrieved by executing an exploratory query corresponding to the interesting faSet that the user has clicked on. An *explanation* is also provided along with the YMAL results, i.e., indicating how these specific recommendations are related to the original query result. Since the number of results of an exploratory query may be large, we rank these results according to an application-dependent *popularity* measure. For the movie database, we rank the results based on the average rating of the movies with which they are related. For example, in (Fig. 3(c)), the directors of the best-rated movies appear first. We present the top-10 recommendations for each faSet. If users wish to do so, they can request to see more recommendations, or click on a different faSet of the original query to get a new exploratory query. They can also exploit the result of the exploratory query to locate interesting faSets in it and get new recommendations related to the exploratory query. Furthermore, users can request the expansion of their original queries. The expansion rules defined by the system administrator are used to expand the original query.

During the demonstration, users will also be able to login to YMALDB as administrators and perform the following administrative tasks: (i) define expansion rules and (ii) see and tune the statistics maintained for the database.

Concerning the first task, administrators are presented with three panels that assist them in defining expansion rules (Fig. 4(a)). To define a new rule $A \rightarrow B$, administrators select from the left panel the attributes in A. Then, they can either let the system suggest a set of expansion attributes B based on the maintained statistics, or select the attributes of B themselves. Afterwards, they can create the new expansion rule by clicking on the buttons shown in the figure. Concerning the second task, we provide a tool that computes and suggests a suitable ϵ approximation value for the maintained statistics given a space allocation limit. Administrators specify a desired target space to be allocated for the statistics and the system shows a plot of ϵ values that will result in database statistics that can fit the allocated space (Fig. 4(b)). Furthermore, to assist administrators in tuning ξ and ξ_0, we provide a tool so that they can select any combination of attributes in the database and see the corresponding faSets for these attributes in the database along with their support. Finally, they can set ϵ, ξ and ξ_0 and press on a button to generate an ϵ-tolerance closed rare faSets summary using ξ and ξ_0 (Fig. 4(c)).

Acknowledgment

This work was co-financed by the European Union (European Regional Development Fund - ERDF) and Greek national funds through the Operational Program "THESSALY - MAINLAND GREECE AND EPIRUS - 2007-2013" of the National Strategic Reference Framework (NSRF 2007-2013).

4. REFERENCES

[1] The Internet Movie Database (IMDb). http://www.imdb.com.
[2] Yahoo!Auto. http://autos.yahoo.com.
[3] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, 2009.
[4] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.*, 31(3), 2006.
[5] J. Cheng, Y. Ke, and W. Ng. δ-tolerance closed frequent itemsets. In *ICDM*, 2006.
[6] M. Drosou and E. Pitoura. ReDRIVE: result-driven database exploration through recommendations. In *CIKM*, 2011.
[7] A. Kashyap, V. Hristidis, and M. Petropoulos. Facetor: cost-driven exploration of faceted query results. In *CIKM*, 2010.
[8] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, 2006.
[9] N. Sarkas, N. Bansal, G. Das, and N. Koudas. Measure-driven keyword-query expansion. *PVLDB*, 2(1), 2009.
[10] K. Stefanidis, M. Drosou, and E. Pitoura. "You May Also Like" results in relational databases. In *PersDB*, 2009.