ReDRIVE: Result-Driven Database Exploration through Recommendations (Extended Version*)

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

Typically, users interact with database systems by formulating queries. However, many times users do not have a clear understanding of their information needs or the exact content of the database, thus, their queries are of an exploratory nature. In this paper, we propose assisting users in database exploration by recommending to them additional items that are highly related with the items in the result of their original query. Such items are computed based on the most interesting sets of attribute values (or faSets) that appear in the result of the original user query. The interestingness of a faSet is defined based on its frequency both in the query result and in the database instance. Database frequency estimations rely on a novel approach that employs an ϵ-tolerance closed rare faSets representation. We report evaluation results of the efficiency and effectiveness of our approach on both real and synthetic datasets.

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

Keywords

Recommendations, Faceted Search, Data Exploration

1. INTRODUCTION

Typically, users interact with a database system by formulating queries. This interaction mode assumes that users are to some extent familiar with the content of the database and also have a clear understanding of their information needs. However, as databases get larger and become accessible to a more diverse and less technically-oriented audience, exploration or recommendation style database interactions seem attractive and useful.

A step towards this direction is offered by facet queries that provide a form of navigational search, where users restrict their results by selecting interesting facets of the original results (e.g., [12]). With facet search, users start with a general query and progressively narrow its results down to a specific item. Other related research includes addressing 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 additional constraints to it (e.g., [17]) to automatically ranking the query results and presenting to the user only the top-k most highly ranked among them (e.g., [7]). The empty-answers problem is commonly handled by relaxing the original query (e.g., [13]).

In this paper, we propose a novel exploration mode of interaction: we present to the users additional items which, although not part of the answer of their original query, may be of interest to them. This way users see information that they may be unaware that exists. For instance, when asking for movies directed by F.F. Coppola, we guide exploration by recommending movies by other directors that have directed movies similar to those of F.F. Coppola, i.e., with similar characteristics, such as, genre or production year. We also consider expanding the original query with additional attributes, by finding correlations with other relations. For example, when asking for the title of a movie, we also look into its genre or other characteristics.

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. The computation of interestingness is based on the frequency of the faSet both in the user query result and in the database instance. Since computing the frequencies of faSets in the database instance on-line has prohibitively high cost, we opt to maintain statistics that allow us to *estimate* those frequencies when needed. More specifically, we propose a novel approach that is based on storing an ϵ-tolerance closed rare faSets representation as a summary of such frequencies and exploit these summaries to estimate the interestingness of the faSets that appear in the result of any given user query.

*This is an extended version of the paper appearing in Proc. of the 20th ACM Conference on Information and Knowledge Management (CIKM 2011), October 24-28, 2011, Glasgow, Scotland, UK.

†Work partially supported by the research program "HRAKLEITOS II" co-funded by the European Union and National Sources.

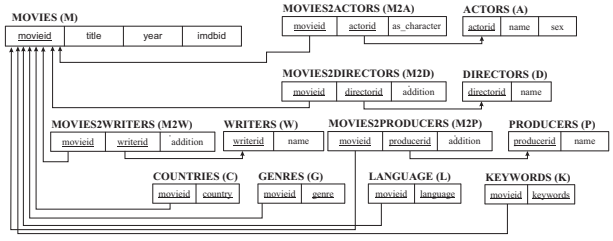


Figure 1: Movies database schema.

We also present a two-phase algorithm for computing the top-k faSets. In the first phase, the algorithm uses the precomputed statistics to set a frequency threshold that is then used to run a frequent itemset based algorithm on the result of the query. We evaluate the performance of our approach using both real and synthetic datasets.

The rest of this paper is organized as follows. Section 2 presents the overall framework, Section 3 its implementation and Section 4 an experimental evaluation. Finally, related work is presented in Section 5 and conclusions are offered in Section 6.

2. THE ReDRIVE FRAMEWORK

Let $\mathcal{D}$ be a relational database with n relations $\mathcal{R} = \{R_1, \ldots, R_n\}$. Let $\mathcal{A}$ be the set of all attributes in $\mathcal{R}$. Without loss of generality, we assume that relation and attribute names are distinct. 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 $sel(Q)$ **AND** $join(Q)$

where $rel(Q)$ is a set of relations, $sel(Q)$ is a conjunction of selection conditions specified by the user, $join(Q)$ is a set of join conditions among the relations in $rel(Q)$ and $proj(Q)$ is the set of projected attributes. For simplicity, we shall focus on *equality* conditions, i.e., $sel(Q) = (A_1 = a_1) \wedge \ldots \wedge (A_m = a_m)$, $m \geq 1$, where $A_i \in \mathcal{A}$ and $a_i \in domain(A_i)$. The *result set*, $Res(Q)$, of a query Q is a relation with schema $proj(Q)$.

Since users must specify in their queries the conditions that the searched items need to satisfy, they must have a relatively clear understanding of the information they are seeking. In this paper, we propose an exploratory way of discovering interesting information based on identifying potentially interesting pieces of information in the result set and then using these pieces of information to explore the database further by recommending additional results to the users.

2.1 Interesting FaSets

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 condition of the form $(A_i = a_i)$, where $A_i \in \mathcal{A}$ and $a_i \in domain(A_i)$. An m-set of facets or m-faSet, $m \geq 1$, is a set of m facet conditions on m different attributes.*

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

```
SELECT D.name, M.title, M.year, G.genre
FROM D, M2D, M, Gs
WHERE D.name = "F. F. Coppola"
      AND D.directorid = M2D.directorid
      AND M2D.movieid = M.movieid
      AND M.movieid = G.movieid;
```

D.name	M.title	M.year	G.genre
F. F. Coppola	The Godfather: Part III	1990	Drama
F. F. Coppola	The Rainmake	1997	Crime
F. F. Coppola	The Godfather	1972	Drama
F. F. Coppola	Rumble Fish	1983	Drama
F. F. Coppola	The Conversation	1974	Thriller
F. F. Coppola	The Outsiders	1983	Drama
F. F. Coppola	Supernova	2000	Thriller
F. F. Coppola	Apocalypse Now	1979	Drama

Figure 2: Example query and result set.

For a faSet f, we use $Att(f)$ to denote its attributes. 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 facets $(A_i = a_i) \in f$. 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 in Figure 1 and the query and its corresponding result set in Figure 2. Then {*G.genre = "Drama"*} or simply {*"Drama"*} is a 1-faSet and {*M.year = "1972"*, *G.genre = "Drama"*} or simply {*"1972"*, *"Drama"*} is a 2-faSet.

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).

Example: Consider the example in Figure 2, where a user poses a query to retrieve movies directed by F.F. Coppola. {*"Drama"*} is a 1-faSet in the result that is likely to interest the user, since it is associated with many of the movies directed by F.F. Coppola. The same holds for the 2-faSet {*"1983"*, *"Drama"*}.

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. For a user information need u_Q expressed through a query Q, let R_{u_Q} be the set of tuples that are relevant to the user need u_Q and $\overline{R_{u_Q}}$ be the set of tuples that are not relevant to u_Q. Then, the relevance score of a faSet f for u_Q is defined as follows:

$$\frac{p(R_{u_Q}|f)}{p(\overline{R_{u_Q}}|f)}$$

where $p(R_{u_Q}|f)$ is the probability that a tuple satisfying f is relevant to u_Q and $p(\overline{R_{u_Q}}|f)$ is the probability that a tuple satisfying f is not relevant to u_Q.

Using the Bayes rule we get:

$$\frac{p(f|R_{u_Q})p(R_{u_Q})}{p(f|\overline{R_{u_Q}})p(\overline{R_{u_Q}})}$$

Since the terms $p(R_{u_Q})$ and $p(\overline{R_{u_Q}})$ have the same value for

all faSets, and thus do not affect their ranking, they can be ignored.

We make the assumption that all relevant to u_Q results are those that appear in $Res(Q)$, thus $p(f|R_{u_Q})$ is equal with the probability that f is satisfied by a tuple in the result set, written $p(f|Res(Q))$. Similarly, $p(f|\overline{R_{u_Q}})$ is the probability that f is satisfied by a tuple that is not relevant, that is, a tuple that does not belong to the result set. We make the logical assumption that the result set is small in comparison with the size of the database, and approximate the non-relevant tuples with all tuples in the database, that is, all tuples in the global relation, denoted by $\mathcal{D}$, with schema $\mathcal{A}$. Based on the above motivation, we provide the following definition for the relevance of a faSet:

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 $Res(Q)$, that is, 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 query. It serves as an indication of how frequent the faSet is in the whole dataset, i.e., it measures the discriminative power of f. Note that when the attributes in $Att(f)$ do not belong to the same relation, to estimate this value we may need to join the respective relations first.

Example: In the example in Figure 2, *"Drama"* appears more frequently than *"Thriller"* in the result set. However, if *"Thriller"* appears only a handful of times in the database, then it would be considered more interesting than *"Drama"*.

In general, a faSet stands out when it appears more frequently in $Res(Q)$ than expected. For a faSet f, $score(f, Q) > 1$, if and only if, its support in the result set is larger than its support in the database, while $score(f, Q) = 1$ means that f appears as frequently as expected.

Clearly, the $sel(Q)$ part of a query is also a faSet. Therefore, another way of interpreting the interestingness score of f for Q is as the confidence of the association rule: $sel(Q) \rightarrow f$. High confidence indicates a strong dependency of the faSet f on the selection conditions of Q.

Finally, note that in particular, for a faSet f with $Att(f) \subseteq Att(sel(Q))$, that is, for a faSet that includes only attributes whose values are specified in the selection conditions, it holds that $score(f, Q) \geq 1$, since $p(f|Res(Q)) = 1$.

2.2 Attribute Expansion

Definition 2 provides a means of ranking the various faSets that appear in the result set $Res(Q)$ of a query Q and discovering the most interesting among them. However, there may be interesting faSets that include attributes not in $proj(Q)$ and thus do not appear in $Res(Q)$. Thus, we would like to extend Definition 2 towards discovering such potentially interesting faSets that include attributes not in $proj(Q)$. This can be achieved by expanding $Res(Q)$ towards other relations in $\mathcal{D}$.

Consider for example the following query that just returns the titles of movies directed by F.F. Coppola:

```
SELECT M.title
FROM D, M2D, M
WHERE D.name = 'F.F. Coppola'
  AND D.directorid = M2D.directorid
  AND M2D.movieid = M.movieid
```

All faSets in the result set of Q will appear once (unless F.F. Coppola has directed more than one movie with the same title). However, including for instance the relation "Countries" in $rel(Q)$ and modifying $join(Q)$ accordingly may disclose interesting information, e.g., that many of the movies directed by F.F. Coppola are related to Romania.

The definition of interestingness is extended to include faSets with attributes not in $proj(Q)$, by introducing an extended query Q' with the same $sel(Q')$ as the original query Q but with additional attributes in $proj(Q')$ and additional relations in $rel(Q')$.

DEFINITION 3 (EXTENDED INTERESTINGNESS SCORE). *Let Q be a query and f be a faSet with $Att(f) \subseteq \mathcal{A}$. The interestingness score of f for Q is equal to:*

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

where Q' is an SPJ query with $proj(Q') = proj(Q) \cup Att(f)$, $rel(Q') = rel(Q) \cup \{R' \mid A_i \in R', \text{for } A_i \in Att(f)\}$, $sel(Q') = sel(Q)$ and $join(Q') = join(Q) \wedge$ (joins with $\{R' \mid A_i \in R', \text{for } A_i \in Att(f)\}$).

2.3 Selecting and Presenting FaSets

As a first step, we identify and present to the user the faSets that have the k highest interestingness scores, where k is an input parameter. In general, the success of recommendations is found to depend heavily on explaining the reasons behind them [21]. Thus, along with each faSet suggestion f, we present an explanation of why f is considered important for the query Q. Such explanations have the general form: "The set of values f appears frequently with the values in $sel(Q)$".

Example: The explanation provided for the interesting 2-faSet {*"1983"*, *"Drama"*} in the example in Figure 2 is: "The year and genre pair (1983, Drama) appears frequently with director F.F Coppola". Assuming that expanding this query towards the "Countries" relation yields the interesting 1-faSet {*"Romania"*}, the corresponding explanation is: "Country Romania appears frequently with director F.F Coppola".

2.4 Exploratory Queries

Besides presenting interesting faSets, we also use faSets to discover interesting pieces of data that are potentially related to the user needs but do not belong to the original query. 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, $sel(Q)$, of Q with equivalent ones, thus allowing new interesting results to emerge. Recall that a high interestingness score for f means that the confidence of $sel(Q) \rightarrow f$ is high, indicating replacing $sel(Q)$ with f, since $sel(Q)$ seems to impose f.

For example, for the interesting faSet {*"Drama"*} in Figure 2, the following exploratory query:

```
SELECT D.name
FROM D, M2D, M, G
```

```
WHERE G.genre = 'Drama'
 AND D.directorid = M2D.directorid
 AND M2D.movieid = M.movieid
 AND M.movieid = G.movieid
```

will retrieve other directors that have also directed drama movies, which is an interesting value appearing in the original query result set.

DEFINITION 4 (EXPLORATORY QUERY). *Let Q be a user query and f be an interesting faSet for Q. The exploratory query $\hat{Q}$ that uses f is an SPJ query with $proj(\hat{Q}) = Attr(sel(Q))$, $rel(\hat{Q}) = rel(Q) \cup \{R' \mid A_i \in R', \text{ for } A_i \in Att(f)\}$, $sel(\hat{Q}) = f \wedge \neg\, sel(Q)$ and $join(\hat{Q}) = join(Q) \wedge$ (joins with $\{R' \mid A_i \in R', \text{ for } A_i \in Att(f)\}$).*

Then, interesting faSets for the exploratory $\hat{Q}$ are recommended to the user. As before, it is central to include an explanation for why each faSet of $\hat{Q}$ is suggested. The explanation in this case specifies that the presented faSet appears often with a value that is very common in the result of the original query Q. For example, assuming that M. Scorsese is a director retrieved by the above exploratory query, then the corresponding explanation would be "Director name M. Scorsese appears frequently with the frequent genre Drama of the original query".

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 follow the various exploratory queries suggested to them to gradually discover other interesting information in the database.

Note that in this paper, for ease of presentation, we have restricted faSets (and the conditions in $sel(Q)$) to attribute value equality. However, the definitions of interestingness and exploratory queries are applicable to any form of attribute value conditions.

Framework Overview: In summary, ReDRIVE database exploration works as follows. Given a query Q, the top-k most interesting faSets for Q are computed and presented to the users. Such faSets may be either interesting pieces (sub-tuples) of the tuples in the result set of Q or extended tuples that include additional attributes not in the original result. Interesting faSets are further used to construct exploratory queries that lead to discovering additional information related to the initial user query. This process can be repeated for each exploratory query.

3. TOP-K FASETS COMPUTATION

In this section, we present algorithms for finding interesting faSets. In particular, first, we consider computing off-line suitable statistics so that we can later estimate $p(f|\mathcal{D})$ for a faSet f. Then, we present an on-line two-phase algorithm for computing the top-k most interesting faSets.

3.1 Estimation of $p(f|\mathcal{D})$

Let Q be a user query with schema $proj(Q)$ and $f = \{A_1 = a_1, \ldots, A_m = a_m\}$ be an m-faSet with $A_i \in proj(Q)$ and $a_i \in domain(A_i)$, $1 \leq i \leq m$. To compute the interestingness of f, according to Definition 2 (and Definition 3), we have to compute two quantities: $p(f|Res(Q))$ and $p(f|\mathcal{D})$.

$p(f|Res(Q))$ is the support of f in $Res(Q)$. This quantity is different for every user query Q and, thus, has to be computed on-line. $p(f|\mathcal{D})$, however, is the same for all user queries.

The straightforward approach is to also compute the value of $p(f|\mathcal{D})$ on-line when required. To do this, we have to execute the following query:

```
SELECT count(*)
FROM rel(Q)
WHERE f AND join(Q)
```

to compute the number of all database tuples satisfying the faSet f. One such query would have to be executed for each examined faSet of $Res(Q)$. As the number of such faSets may be large, the cost of executing these queries on-line may become prohibitively large. Such an approach is feasible only in the case where a handful of faSets need to be examined. However, this is not a typical case, since many faSets are present even in relatively small query results. Therefore, it is necessary to maintain some form of information (or statistics) that will allow us to have an estimation of $p(f|\mathcal{D})$ without needing to execute such queries at execution time. Next, we show how we can maintain such information.

3.1.1 Full Statistics

Let m_{max} be the maximum number of projected attributes of any user query, i.e., $m_{max} = |\mathcal{A}|$. An exhaustive approach to statistics maintenance would be to generate all possible faSets of size up to m_{max} and pre-compute their support in $\mathcal{D}$. Such an approach, however, is infeasible even for small databases due to the combinatorial amount of possible faSets. Consider a database with a single relation R containing 10 attributes. If each attribute has on average 50 distinct attribute values, then R contains, for example, $7.875 \cdot 10^{10}$ faSets of size 5. Clearly, it is not possible to store and maintain information about the support of all possible faSets.

3.1.2 Partial Statistics

A feasible and efficient solution to our problem must reach a compromise between the on-line computation of $p(f|\mathcal{D})$ and the maintenance of full statistics. There are a number of available options for this which we examine in the following.

Maintaining 1-faSets.

A first approach is to pre-compute and store the support of each faSet of size one that appears in the database. Then, assuming that facet conditions are satisfied independently from each other, the support of a higher-order m-faSet is equal to:

$$p(f|\mathcal{D}) = p(\{A_1 = a_1, \ldots, A_m = a_m\}|\mathcal{D}) = \prod_{i=1}^{m} p(\{A_i = a_i\}|\mathcal{D})$$

This approach requires the storage of information for only a relatively small number of faSets, i.e., $\sum_{A_i \in \mathcal{A}} |domain(A_i)|$ faSets. However, although commonly used in the literature, the independence assumption is quite strong for real-data applications where it rarely holds in practice.

Maintaining faSets of size up to ℓ.

Considering that we are able to afford some extra storage space to maintain the support of more faSets, say up to size ℓ, we can have a more accurate estimation of the support of a high-order faSet f with size m, $m > \ell$, using some more sophisticated method.

One such method that can be applied, similarly to [17], is Iterative Proportional Fitting (IPF) [4]. Let $f = \{A_1 = a_1, \ldots, A_m = a_m\}$ be a faSet with size m, $m > \ell$. f

director = Coppola	year = 2010	genre = Action	probability
0	0	0	p_1
0	0	1	p_2
0	1	0	p_3
0	1	1	p_4
1	0	0	p_5
1	0	1	p_6
1	1	0	p_7
1	1	1	p_8

$p_5 + p_6 + p_7 + p_8 = p(\{director = Coppola\})$
$p_3 + p_4 + p_7 + p_8 = p(\{year = 2010\})$
$p_2 + p_4 + p_6 + p_8 = p(\{genre = Action\})$
$p_7 + p_8 = p(\{director = Coppola, year = 2010\})$
$p_6 + p_8 = p(\{director = Coppola, genre = Action\})$
$p_4 + p_8 = p(\{year = 2010, genre = Action\})$
$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 1$

Figure 3: IPF constraints.

can be viewed as the result of a probabilistic experiment as follows: We associate with each facet condition $(A_i = a_i)$ in f a binary variable. This binary variable denotes whether the corresponding facet condition is satisfied or not. This experiment has $v = 2^m$ possible outcomes.

Let p_1 be the probability that the outcome is $(0, 0, \ldots, 0)$, p_2 be the probability that the outcome is $(0, 0, \ldots, 1)$ and so on. That is, p_i is the probability of f being satisfied by exactly the conditions corresponding to the variables equal to 1 as specified by the i^{th} possible outcome, $1 \leq i \leq v$ (see Figure 3 for an example with $m = 3$). Having pre-computed the support of low-order faSets, we have some knowledge for the values of the discrete distribution $\mathbf{p} = (p_1, \ldots, p_v)^T$: First, all p_is for which a faSet f of size m with $m \leq \ell$ is satisfied must sum up to $p(f|\mathcal{D})$, i.e., the pre-computed support. Second, all p_is must sum up to 1. For example, for $\ell = 2$, we have m constraints due to the pre-computed support values of all 1-faSets and $m(m-1)/2$ constraints due to the 2-faSets. Therefore, we have $m + m(m-1)/2 + 1$ constraints in total.

However, there are more variables p_i than constraints, therefore we cannot determine all values of $\mathbf{p}$. Iterative Proportional Fitting (IPF) can be employed to estimate $\mathbf{p}$. This method is based on the Principle of Maximum Entropy, which states that, since there is no reason to bias the estimated distribution of $\mathbf{p}$ towards any specific form, then the estimation should be as close to the uniform distribution as possible. IPF initializes the elements of $\mathbf{p}$ randomly and then iteratively checks each available constraint and scales by an equal amount the elements of $\mathbf{p}$ participating in the constraint so that the constraint is satisfied. It can be proved that this process converges to the maximum entropy distribution.

Example: Let R be a relation with a number of attributes including *director*, *year*, *genre* and that $\ell = 2$, i.e., we know the support of all faSets of size 1 and 2, e.g., {*director* = "*F.F. Coppola*"}, {*year* = "*2010*"}, {*year* = "*2010*", *genre* = "*Action*"} and so on. To estimate the support of the 3-faSet {*director* = "*F.F. Coppola*", *year* = "*2010*", *genre* = "*Action*"}, we have $m = 3$, $v = 8$ and $m + m(m-1)/2 + 1 = 7$ constraints, as shown in Figure 3.

Maintaining ϵ-Tolerance Closed Rare FaSets.

The method described above can estimate the support of high-order faSets, while maintaining information only about a relatively small amount of low-order ones. However, the maximum entropy criterion smooths the estimated distribution. This may lead to the loss of interesting information. Consider for example that the faSets {*G.genre* = "*Sci-Fi*"}, {*M.year* = "*2000*"}, {*M.year* = "*2005*"} have similar supports, while the supports of {*G.genre* = "*Sci-Fi*", *M.year* = "*2000*"} and {*G.genre* = "*Sci-Fi*", *M.year* = "*2005*"} differ a lot. IPF (for $\ell = 1$) will estimate similar values for these two faSets. Therefore, we propose a different form of statistics, aiming at capturing such fluctuations in the support of related faSets. To do this, we employ the notion of δ-tolerance frequent itemsets [8] and define ϵ-tolerance closed rare faSets as explained in following.

Background Definitions: In data mining, the term *itemset* refers to a set of items. An itemset is said to be *frequent* in a dataset if its frequency is above a specific threshold. Otherwise it is called *rare*. We observe that there is a one-to-one correspondence between itemsets and faSets. An itemset of size m containing the items $\{A_1 = a_1, \ldots, A_m = a_m\}$ appears in exactly the same set of tuples that satisfy the m-faSet $f = \{A_1 = a_1, \ldots, A_m = a_m\}$. Therefore, we can say that a faSet f is frequent (FF) for a set of tuples S if its support in S is above a specific threshold. Otherwise it is called rare (RF).

Based on related itemset definitions from data mining, we call a faSet f *closed frequent* (CFF) for S if it is frequent and has no proper superset f', such that, f' has the same support as f in S, where f' is called a proper superset of f if $f \subset f'$, i.e., every item of f is contained in f' but there is at last one item in f' that is not contained in f. We also say that f is a proper subset of f'. Similarly, we define a faSet f to be *closed rare* (CRF) for S if it is rare and has no proper subset f', such that, f' has the same support as f in S. Finally, we say that a faSet f is *maximal frequent* (MFF) for S if it is frequent for S and has no superset f' such that f' is frequent for S and, finally, a faSet f is *minimal rare* (MRF) for S if it is rare and has has no subset f' such that f' is rare for S.

Statistics based on ϵ-tolerance: Maintaining the support of a number of representative faSets can assist us in estimating the support of a given faSet f. There are many available options regarding whether we maintain information about frequent or rare faSets and also the granularity of the maintained statistics.

Generally, faSets that appear frequently in the database $\mathcal{D}$ are not expected to be interesting, even if they appear often in the result of user queries, since this is expected. Therefore, it would be more useful to maintain information about the frequency of rare faSets in $\mathcal{D}$. We use $count(f, S)$ to denote the absolute number of appearances of a faSet f in a set of tuples S.

If we maintain the MRFs, we can derive all corresponding RFs but not their actual support, while if we keep the CRFs we can retrieve these supports as well. Generally, the number of CRFs lies between the number of RFs and that of MRFs. However, since according to the definition of CRFs any RF having a distinct support is a CRF, the number of CRFs can be very close to that of the RFs and, therefore, it is impractical to store as previously discussed.

In our approach, we employ the notion of δ-tolerance closed frequent itemsets and maintain statistics that allow us to store a tunable amount of faSets from which we will be able to retrieve a bounded estimation of the support of a given faSet in the database. A frequent itemset is called δ-tolerance closed if it has no frequent proper super-itemset

such that the support of the two itemsets differ more than a threshold related to a constant δ. Similarly, we define ϵ-*tolerance closed rare faSets* (ϵ-CRFs) as follows:

DEFINITION 5 (ϵ-CRF). *A faSet f is called ϵ-CRF for a set of tuples S, if and only if, it is rare for S and it has no proper immediate rare subset f', i.e., $|f'| = |f| - 1$, such that, $count(f', S) \leq (1 + \epsilon)\ count(f, S)$, where $\epsilon \geq 0$.*

Intuitively, a rare faSet f is an ϵ-CRF if, even if we increase its count by a constant ϵ, all its subsets still have a larger frequency than f. This means that f has a different frequency from all its subsets and cannot be estimated (or represented) by any of them.

Let us assume that a set of ϵ-CRFs is maintained for some value of ϵ. We denote this set C. An RF f either belongs to C or not. If $f \in C$, then the support of f is stored and its count is readily available. If not, then, according to Definition 5, there is some subset of f that belongs to C whose support is close to that of f. Therefore, given an RF f, we can estimate its count based on its closest subset in C. If there are many such subsets, we use the one with the minimum count, since this can estimate the count of f more accurately. We use $C(f)$ to denote the faSet in C that is the most suitable one to estimate the count of f. The following lemma holds:

LEMMA 1. *Let C be a set of ϵ-CRFs for a set of tuples S and f be a faSet, $f \notin C$. Then, there exists f', $f' \in C$, such that, $count(f', S) \leq \phi\ count(f, S)$, where $\phi = (1 + \epsilon)^{(|f|-|f'|)}$.*

In order to provide estimations, each ϵ-CRF is stored along with its *frequency extension*. The frequency extension of an ϵ-CRF is defined as follows:

DEFINITION 6 (FREQUENCY EXTENSION). *Let C be a set of ϵ-CRFs for a set of tuples S and f be a faSet in C. Let also $\mathcal{X}(f)$ be the set of all RFs represented in C by f. Then, $X_i(f) = \{x | x \in \mathcal{X}(f) \wedge |x| - |f| = i\}$, $1 \leq i \leq m$, where $m = \max\{i | X_i(f) \neq \emptyset\}$. The frequency extension of f for i, $1 \leq i \leq m$, is defined as:*

$$ext(f, i) = \frac{\sum_{x \in X_i(f)} \frac{count(x,S)}{count(f,S)}}{|X_i(f)|}$$

Intuitively, the frequency extension of f for i is the average count difference between f and all the faSets that f represents whose size difference from f is equal to i. Given a faSet f, the estimation of $p(f|\mathcal{D})$, denoted $\tilde{p}(f|\mathcal{D})$ is equal to:

$$\tilde{p}(f|\mathcal{D}) = count(C(f), S) \cdot ext(C(f), |f| - |C(f)|)$$

It holds that:

LEMMA 2. *Let f be an ϵ-CRF. Then, $\forall i$, it holds that $\frac{1}{\phi} \leq ext(f, i) \leq 1$, where $\phi = (1 + \epsilon)^i$.*

PROOF. At one extreme, all faSets in $X_i(f)$ have the same count as f. Then, $\forall x \in X_i(f)$ it holds that $count(x, S) = count(f, S)$ and $ext(f, i) = 1$. At the other extreme, all faSets in $X_i(f)$ differ as much as possible from f. Then, $\forall x \in X_i(f)$ it holds that $count(f, S) = \phi\ count(x, S)$ and $ext(f, i) = 1/\phi$. □

Similarly to the proof in [8], it can be shown that the estimation error is bounded by ϕ, i.e., by ϵ. More specifically, let f be an RF and $|f| - |C(f)| = i$. The estimation error for $p(f|\mathcal{D})$ is bounded as follows:

$$\frac{1}{\phi} - 1 \leq \frac{\tilde{p}(f|\mathcal{D}) - p(f|\mathcal{D})}{p(f|\mathcal{D})} \leq \phi - 1$$

The proof is omitted.

3.2 The Two-Phase Algorithm

Given a query Q, our goal is to locate the k faSets with the highest interestingness scores. We consider first faSets in the result set and defer the treatment of extended faSets until the next section. Clearly, the brute-force method of generating all possible faSets in $Res(Q)$ and computing their score is exponential on the number of distinct values that appear in $Res(Q)$. Applying an a-priori approach for generating and pruning faSets is not applicable either, since *score* is neither an upwards nor a downwards measure, as shown below. Recall that, a measure d is *monotone* or *upwards closed* if for any two sets S and S', $S \subseteq S' \Rightarrow d(S) \leq d(S')$ and *anti-monotone* or *downwards closed* if $S \subseteq S' \Rightarrow d(S) \geq d(S')$.

PROPOSITION 1. *Let Q be a query and f be a faSet. Then, $score(f, Q)$ is neither an upwards nor a downwards closed measure.*

PROOF. Consider a database consisting of a single relation R with three attributes A, B and C and three tuples $\{\{a_1, b_1, c_1\}, \{a_1, b_1, c_2\}, \{a_1, b_2, c_1\}\}$. Let $Res(Q) = \{\{a_1, b_1, c_1\}, \{a_1, b_2, c_1\}\}$ and the following three faSets: $f_1 = \{A = a_1\}$, $f_2 = \{A = a_1, B = b_1\}$, $f_3 = \{A = a_1, B = b_1, C = c_1\}$. For f_2, there exists both a subset (f_1) and a superset (f_3) with larger scores than it. □

This implies that we cannot employ any subset or superset relations among the faSets of $Res(Q)$ to prune the search space.

Algorithm

As a baseline approach to reduce the number of examined faSets of $Res(Q)$, we consider only the most frequent faSets of $Res(Q)$, motivated by the fact that faSets that appear in $Res(Q)$ frequently are likely to be highly interesting to the user. To this end, we apply an adaptation of a frequent itemset mining algorithm [11], such as Apriori or FP-Growth, to generate all frequent faSets of $Res(Q)$, that is, all faSets with support larger than some pre-specified threshold $minsupp_f$. Then, for each frequent faSet f, we use the maintained statistics to estimate $p(f|\mathcal{D})$ and compute $score(f, Q)$.

This baseline approach has the problem of being highly dependent on $minsupp_f$. A large value of $minsupp_f$ may lead to losing some less frequent in the result but very rarely appearing in the dataset faSets, whereas a small value may result in a very large number of candidate faSets being examined. Therefore, we propose a Two-Phase Algorithm (TPA), described next, that addresses this issue by setting $minsupp_f$ to an appropriate value so that all top-k faSets are located without generating redundant candidates. TPA assumes that the maintained statistics are based on keeping rare faSets of the database $\mathcal{D}$. Let $minsupp_r$ be the support threshold of the maintained rare faSets.

Algorithm 1 Two-Phase Algorithm (TPA).

Input: Q, $Res(Q)$, k, C, $minsupp_r$ of C.
Output: The top-k interesting faSets for Q.

```
1: begin
2: S ← ∅
3: A ← all 1-faSets of Res(Q)
4: for all faSets f ∈ C do
5:    if all 1-faSets g ⊆ f are contained in A then
6:       f.score = score(f, Q)
7:       S ← S ∪ {f}
8:    end if
9: end for
10: for all tuples t ∈ Res(Q) do
11:    generate all faSets f ⊆ t, s.t. ∃ g ∈ S with g ⊆ f
12:    for all such faSets f do
13:       f.score = score(f, Q)
14:       S ← S ∪ {f}
15:    end for
16: end for
17: minsupp_f ← (k^th highest score in S) × minsupp_r
18: candidates ← frequentFaSetMiner(Res(Q), minsupp_f)
19: for all faSets f in candidates do
20:    f.score = score(f, Q)
21:    S ← S ∪ {f}
22: end for
23: return The k faSets in S with the highest scores
24: end
```

In the first phase of the algorithm, all facet conditions, or 1-faSets, that appear in $Res(Q)$ are located. TPA checks which rare faSets of $\mathcal{D}$, according to the maintained statistics, contain only facet conditions from $Res(Q)$. Let X be the set of faSets. Then, in one pass of $Res(Q)$, all faSets of $Res(Q)$ that are supersets of some faSet in X are generated and their support in $Res(Q)$ is measured. For each of the located faSets, $score(f, Q)$ is computed. Let s be the k^{th} highest score among them. TPA sets $minsupp_f$ equal to $s \times minsupp_r$ and proceeds to the second phase where it executes a frequent itemset mining algorithm with threshold equal to $minsupp_f$. It is easy to see that any faSet in $Res(Q)$ less frequent than $minsupp_f$ clearly has score smaller than the k^{th} faSet located in the first phase and thus can be safely ignored. To see this, let f be a faSet examined in the second phase of the algorithm. Since the score of f has not been computed in the first phase, then $p(f|\mathcal{D}) > minsupp_r$. Therefore, for $score(f, Q) > s$ to hold, it must be that $p(f|Res(Q)) > s \times p(f|\mathcal{D})$, i.e., $p(f|Res(Q)) > s \times minsupp_r$. TPA is shown in Algorithm 1, where we use C to denote the collection of maintained statistics.

3.3 FaSet Expansion

For a query Q, besides the faSets whose attributes belong to $proj(Q)$, we would also like to consider potentially interesting faSets that have additional attributes. Clearly, considering all possible faSets for all combinations of (attribute, value) pairs is prohibitive. Instead, we consider adding to $proj(Q)$ a few additional attributes $\mathcal{B}$ that appear relevant. Then, besides Q, we construct and execute Q' as defined in Definition 3 and use TPA to compute the top-k (expanded) faSets of Q'.

Our goal is to select a set of attributes $\mathcal{B}$, $\mathcal{B} \subset \mathcal{A}$ and $\mathcal{B} \cap proj(Q) = \emptyset$, such that the expected value of $p(f'|res(Q'))$ is large. The first approach we consider is selecting the attributes $B_i \in \mathcal{B}$ based solely on the attributes that appear in Q. In particular, for each A_i, we maintain a list of promising attributes for expansion and use them, each time A_i appears in $proj(Q)$ or $sel(Q)$. This is done in the level of relations. Let R be a relation appearing in $rel(Q)$. We add a new relation R' in $rel(Q)$ and a pre-specified attribute B_i of R' in $proj(Q)$.

A more expensive but potentially more accurate approach is to consider $sel(Q)$ in selecting the attributes. In this case, we need to maintain statistics for frequent faSets in $\mathcal{D}$ similar to those maintained for rare faSets. We look into the frequenet faSets and select a frequent faSet $f' \supset sel(Q)$. Then, we add the additional attributes of f' in $proj(Q)$.

4. EXPERIMENTAL RESULTS

In this section, we present experimental results of the deployment of our approach. We use both real and synthetic datasets. Synthetic datasets consist of a single relation with five attributes and 1000 tuples, each of which takes values from a zipf distribution with parameter θ ("`ZIPF-`θ") and 5 distinct values. We also use two real databases. The first one ("`AUTOS`") is a single-relation database consisting of 41 characteristics for 15,191 used cars from Yahoo!Auto [2]. The second one ("`MOVIES`") is a multi-relation database containing information extracted from the Internet Movie Database [1]. The schema of this database is shown in Figure 1. The cardinality of the various relations ranges from around 10,000 to almost 1,000,000 tuples. We consider a subset of relations, namely Movies, Movies2Directos, Directors, Genres and Countries.

We use MySQL 5.0 to store our data. Our system is implemented in JDK 1.6 and connects to the DBMS through JDBC. Our experiments were executed on an Intel Pentium Core2 2.4GHz PC with 2GB of RAM.

4.1 Performance Evaluation

We start by presenting performance results. There are two building blocks for ReDRIVE. The first is a precomputation step that involves maintaining statistics for estimating the frequency of the various faSets in the database. The second involves the run-time deployment of the maintained statistics in conjunction with the results of the user query towards discovering the k-most interesting faSets.

Statistics Generation: First, we evaluate the various options for maintaining statistics in terms of (i) storage, measured as the number of maintained faSets and (ii) generation time. We consider first maintaining ϵ-CRFs, as proposed in Section 3, and report results for all the intermediate computation steps, i.e., locating MRFs, RFs and CRFs. We base our implementation for locating MRFs and RFs on the MRG-Exp and Arima algorithms [20] and use an adapted version of the CFI2TCFI algorithm [8] for producing ϵ-CRFs. For a given $minsupp_r$, all MRFs are located. However, locating all respective RFs, for large datasets becomes inefficient, due to the exponential nature of algorithms such as Apriori. To overcome this, we use a *random walk* based approach [10] to generate RFs. In particular, we do not produce all RFs as an intermediate step for computing ϵ-CRFs but, instead, we produce only a subset of them discovered by random walks initiated at the MRFs. Never-

Table 1: Number of FaSets.

$minsupp_r$	# MRFs	# RFs	# CRFs	# ϵ-CRFs				
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$
ZIPF-1.0								
5%	186	2983	2283	2283	2177	1319	778	243
10%	145	3402	2639	2639	2505	1541	881	254
20%	54	3238	2547	2547	2443	1556	876	154
ZIPF-0.5								
5%	197	3963	3329	3329	3263	2516	1578	246
10%	248	4284	3649	3649	3574	2811	1700	292
20%	55	4026	3401	3401	3339	2640	1663	118
AUTOS								
5%	2106	2960	2737	2715	2683	2617	2547	2446
10%	2014	2856	2606	2591	2561	2486	2437	2338
20%	1960	2654	2421	2398	2365	2278	2231	2126
MOVIES								
5%	11533	11809	11787	11787	11783	11757	11730	11630
10%	11537	11820	11803	11803	11798	11768	11734	11621
20%	11538	11826	11802	11802	11799	11776	11750	11643

(a)

# 1-faSets	# 2-faSets
ZIPF-1.0	
25	250
ZIPF-0.5	
25	250
AUTOS	
17664	851954
MOVIES	
67094	561223

(b)

Table 2: FaSet Generation Time (in ms).

$minsupp_r$	MRFs	RFs	CRFs	ϵ-CRFs				
				$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.5$	$\epsilon = 0.7$	$\epsilon = 0.9$
ZIPF-1.0								
5%	1344	54719	4641	3500	3437	3516	3516	3562
10%	860	243000	6156	4625	4578	4556	4703	4719
20%	281	58766	5610	4265	4250	4296	4359	4391
ZIPF-0.5								
5%	1344	67031	8328	6750	6765	6828	6890	7000
10%	1391	76188	10157	8188	8218	8375	8453	8563
20%	250	70016	8969	7203	7281	7360	7391	7687
AUTOS								
5%	86125	4564641	2969	2610	2578	2547	2571	2531
10%	49813	4061344	2500	1938	1953	1953	1953	1953
20%	21984	3770234	2187	1797	1796	1797	1781	1781
MOVIES								
5%	21844	2225890	6781	6969	7000	7563	7500	7422
10%	22953	2218250	6890	7156	7203	7063	7390	7093
20%	21468	2164234	6703	6907	6907	6890	6890	6906

theless, our results indicate that, even though not all RFs are generated, we still achieve good estimations.

Table 1(a) shows the number of the produced faSets for our datasets for different values of $minsupp_r$ and ϵ. The number of the produced RFs and the complexity of their generation depends also on the number of random walks. In the reported results, we kept the number of random walks fixed (equal to 20). Clearly, given enough time and memory, a larger number of random walks can be used to produce more "complete" statistics. As ϵ increases, an ϵ-CRF is allowed to represent faSets with larger support difference and, thus, the number of maintained statistics decreases. Also, as $minsupp_r$ increases, more faSets of the database are considered to be rare and, thus, the size of statistics becomes larger. In our experiments, this is not always the case because of the random walks technique employed to retrieve RFs. Note also that the number of ϵ-CRFs is much smaller than the number of RFs, even for values of ϵ as low as 0.5. For comparison, we also report the number of 1-faSets and 2-faSets (Table 1(b), we excluded id attributes) that could alternatively be used for estimating frequencies, as discussed in Section 3, which is considerably large for the real databases.

Table 2 reports the execution time to generate the statistics. We break down the execution time into three stages: (i) the time required to locate all MRFs, (ii) the time required to generate RFs based on the MRFs and (iii) the time required to extract the final ϵ-CRFs based on all RFs. The main overhead is induced by the stage of generating RFs of the database. We do not report the execution time for locating 1-faSets and 2-faSets. This time was trivial for the synthetic datasets due to the small domain size of each attribute and very long for the real datasets; in which case we had to execute the computation in batches because of memory limitations.

In particular, the MOVIES database has a large number of rare 1-faSets that appear only once in the database. To avoid generating all super-sets of such 1-faSets, we use the following approach. In a single scan of the dataset, we identify such 1-faSets, insert them in a hash-based data structure (in particular, a Bloom filter) and delete them from the database. Then, we proceed with the generation of MRFs.

Table 3: Average absolute estimation error.

$minsupp_r$	m	ϵ				
		0.1	**0.3**	**0.5**	**0.7**	**0.9**
ZIPF-1.0						
5%	2	0	0	0	0	0
	3	0	0	5.20	4.80	2.67
	4	0	0.14	2.33	3.47	4.30
	5	0	4.00	1.66	1.77	1.43
10%	2	0	0	0	0	0
	3	0.23	3.17	3.50	7.36	8.03
	4	1.25	1.70	2.45	3.58	1.41
	5	1.00	1.50	1.03	1.75	1.30
20%	2	0	0	0	8.10	13.59
	3	2.30	2.37	4.19	7.85	4.26
	4	3.40	1.80	1.89	1.47	2.32
	5	1.00	1.93	2.80	2.30	1.90

Estimation Accuracy: Next, we evaluate how well ϵ-CRFs estimate the support of a random rare faSet in the database. To do this, we employ our synthetic datasets and randomly construct a number of faSets for each of them. More specifically, for each relation R, we generate random faSets of length $1, \ldots, 5$. Then, we probe our statistics to retrieve estimations for the frequency of 10 such rare faSets for each size. Table 3 shows the average absolute estimation error for the `ZIPF-1.0` dataset. We observe that, even though we do not have the complete set of ϵ-CRFs available, because of our random walk approach to producing RFs, the estimation error remains low. Similar results are attained for the `AUTOS` and the `MOVIES` databases, where the absolute estimation error is higher than for the synthetic datasets but still relatively low. For example, the average absolute estimation error for the `AUTOS` database is around 40 (the higher value is due to the larger interval of faSets support values than in the synthetic datasets).

We also experimented with using IPF for the estimation. However, this approach turned out to be very inefficient for estimating the frequency of rare faSets. This was mainly due to two reasons. Take for example 3-faSets. On one hand, the estimations attained for rare 3-faSets that consist of rare 2-faSets were very small (most often equal to zero) due to the maximum entropy principle that tends to considers faSet co-occurrences independent. For example, for the faSets {*make* = "*Lexus*", *state* = "*VA*", *navigation_system* = "*Yes*"} and {*make* = "*Ford*", *state* = "*FL*", *alarm* = "*No*"} of the `AUTOS` database, the estimated support was zero while there are 20 and 80 tuples, respectively, in the database. On the other hand, the estimations attained for rare 3-faSets that consist of frequent 2-faSets resulted in over-estimations (often ten-fold).

Top-k FaSet Discovery: Next, we compare the baseline and the Two-Phase algorithms described in Section 3. For the `MOVIES` dataset, the TPA is slightly modified to take into consideration the special treatment of rare 1-faSets with frequency 1. In this case, we first look-up the hash-based structure for any supersets of the 1-faSets of $Res(Q)$. In case such subsets exist, then the corresponding 1-faSets of $Res(Q)$ and all their supersets in $Res(Q)$ are higher ranked than any other faSet.

For the synthetic datasets, we generate random queries to test our algorithms, while for the `AUTO` and `MOVIES` databases we use the example queries shown in Figures 5a and 5b, respectively. These queries were selected so that their result set included various combinations of rare and frequent faSets. Figure 4 shows the 1st and 20th highest ranked interestingness score retrieved, i.e., for TPA we set $k = 20$ and for the baseline approach we start with a high $minsupp_f$ and gradually decrease it until we get at least 20 results. We see that TPA is able to retrieve more interesting faSets, mainly due to the first phase where rare faSets of $Res(Q)$ are examined.

We set $k = 20$ and $minsupp_r = 10\%$ and experimented with various values of ϵ. We saw that ϵ did not affect the interestingness scores of the top-k results considerbaly. For the above reported results ϵ was equal to 0.5. In all cases except for q_3 of the `AUTOS` database, TPA located k results during phase one and, thus, phase two was never executed. This means that in all cases there were some faSets present in $Res(Q)$ that were quite rare in the database and, thus, their interestingness was high.

The efficiency of TPA depends on the size of $Res(Q)$, since in phase one the tuples of $Res(Q)$ are examined for locating supersets of faSets in the statistics. However, TPA was very efficient for result sizes up to a few hundred results, requiring from under a second to around 5 seconds to run.

4.2 Exploring the two Real Databases

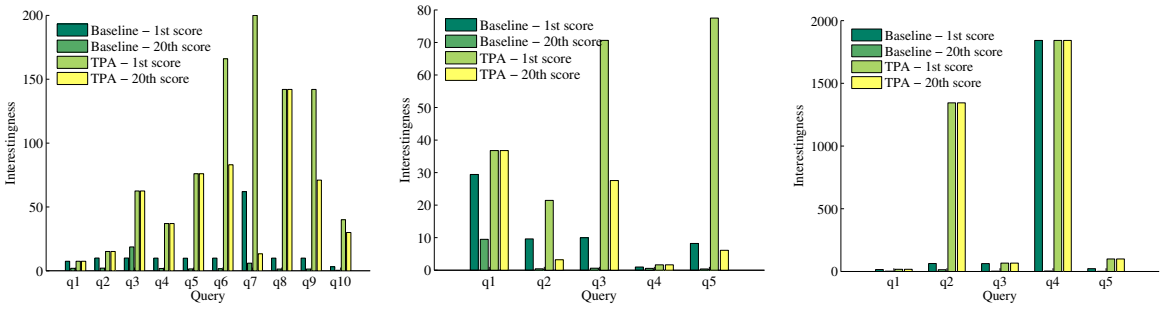
Finally, we present results of our exploration of the `AUTO` and `MOVIES` databases, using the example queries of Figures 5a and 5b respectively. When expanding queries, we considered expansions towards predefined attributes based on $proj(Q)$. Due to space limitations, we next make some interesting observations about the acquired results.

Take for example q_1 of the `AUTOS` database which is a query whose result set includes many rare faSets, all having a high interestingness score, e.g., {*make* ="*Land Rover*", *name* = "*Land Rover Discovery II HSE7*"} and {*make* = "*Mercedes*", *name* = "*Mercedes-Benz G55 AMG)*"}. Expanding q_1 towards the "state" attribute reveals other interesting faSets with varying scores, such as {*make* = "*Land Rover*", *name* = "*Land Rover Range Rover*", *state* = *VA*} and {*make* ="*Cadillac*", *state* = *DE*}, i.e., another Land Rover model was ranked higher because of "state" and also Cadillacs appeared, suggesting that such combinations are highly related with navigation systems (which appear in $sel(q_1)$).

Another interesting example is q_2 for the `MOVIES` database where the highly interesting faSet {*country* ="*Switzerland*", *genre* = "*Sci-Fi*"} is retrieved. This faSet is rare in $Res(q_2)$ but is also extremely rare in the database (appearing just 4 times), so it is very interesting that it was located by the user query. Another highly ranked faSet is {*country* ="*Romania*"} (and its supersets) which has a high interestingness score since it appears relatively frequently in $Res(q_2)$ while it is not a very frequent value in the database, which is generally dominated by USA productions.

5. RELATED WORK

In this paper, we have proposed a novel database exploration model. A common exploration technique is faceted search (e.g., [15, 12, 9]), where results of a query are classified into different multiple categories or facets and the user refines these results by selecting one or more facet condition. Our approach is different in that we do not tackle refinement. Our goal is to identify faSets, possibly expand



(a) ZIPF-1.0 (b) AUTOS (c) MOVIES

Figure 4: Baseline vs. TPA.

q_1: **select** make, name
from autos
where navigation_system = 'Yes';

q_2: **select** make, air_condition, alarm
from autos
where state = 'FL';

q_3: **select** make, name, sunroof
from autos
where state = 'MD' **and** make = 'Lexus';

q_4: **select** make, state, spoiler
from autos
where air_condition = 'Yes' **and** power_steering = 'Yes';

q_5: **select** make, state, side_air_bag
from autos
where child_safety = 'Yes' **and** cruise_control = 'Yes';

(a) AUTOS.

q_1: **select** D.name, G.genre, M.year
from C, M, D, M2D, G
where *join* **and** C.country='France';

q_2: **select** C.country, G.genre, M.year
from C, M, D, M2D, G
where *join* **and** D.name='Coppola, Francis Ford';

q_3: **select** C.country, M.year
from C, M, D, M2D, G
where *join* **and** M2D.addition='Uncredited';

q_4: **select** D.name, G.genre
from C, M, D, M2D, G
where *join* **and** D.name='Coppola, Francis Ford' **and** M2D.addition='Uncredited';

q_5: **select** D.name, C.country
from D, M2D, M, C
where *join* **and** M.year=2000;

(b) MOVIES.

Figure 5: Dataset queries.

them and then use them to discover other interesting results not part of the original query.

There is also some relation with *query reformulation*. In this case, a query is relaxed or restricted when the number of results of the original query are too few or too many respectively using term rewriting or query expansion to increase recall and precision of the original query (e.g., [17]). Again our aim is to present interesting results that are not part of the original query but are highly related to them. Note that we base the computation of interestingness for our results on the surprise value. Similar measures have been used in the literature, such as unexpectedness [24], χ^2 [5] and other variants. Besides restricting the query, another common method of addressing the too-many answers problem is ranking the results of a query and presenting only the top-k most highly ranked ones to the user. This line of research is extensive; the work most related to ours is research based on automatically ranking the results [7, 3]. Besides addressing a different problem, our approach is also different in that the granularity of ranking in our approach is in the level of faSets as opposed to whole tuples. We also propose a novel method for frequency estimation that makes no independence assumptions.

Yet another method of exploring results relies on *why queries* that consider the presence of unexpected tuples in the result and *why not* queries that consider the absence of expected tuples in the result. For example, ConQueR [22] proposes posing follow-up queries for *why not* by relaxing the original query. In our approach, we find interesting faSets in the result based on their frequency and other faSets highly correlated with them.

In some respect, exploration queries may be seen as recommendations. Extending database queries with recommendations has been suggested in two recent works, namely [14] and [6]. [14] proposes a general framework and a related engine for the declarative specification of the recommendation process. Our recommendations here are of a very specific form. Recommendations in [6] are based on the past behavior of similar users, whereas we consider only the content of the database and the result. Some initial ideas of our work can be found in [19].

A somewhat related problem is finding interesting or *exceptional* cells in an OLAP cube [16]. These are cells whose actual value differs substantially from the anticipated one. The anticipated value for a cell is estimated based on the values of its adjacent cells at all levels of group-bys. The techniques used in that area are different though and no additional items are presented to the users.

Finally, another related problem is constructing a query whose execution will yield results equivalent to a given result set [23, 18]. Our work differs in that we do not aim at constructing queries but rather guiding the users towards related items in the database that they may be unaware of.

6. CONCLUSIONS

In this paper, we introduced ReDRIVE, a novel database exploration framework based on presenting to the users additional items which may be of interest to them although not part of the results of their original query. The computation of such results is based on identifying the most interesting sets of (attribute, value) pairs, or faSets, that appear in the result of the original user query. The computation of interestingness is based on the frequency of the faSet in the user query result and in the database instance. Besides proposing a novel mode of exploration, other contributions of this

work include (i) a frequency estimation method based on storing an ϵ-tolerance closed rare faSets representation and (ii) a two-phase algorithm for computing the top-k faSets.

There are many directions for future work. One is to extend our work to more general types of facet conditions. Although the general framework is readily applicable, this will require extending our methods for maintaining statistics. Another possible direction is to consider that a history of previous database queries and results is available and extend our work to include faSets that appear in this history.

7. REFERENCES

[1] The Internet Movie Database (IMDb). http://www.imdb.com.

[2] Yahoo!Auto. http://autos.yahoo.com.

[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[4] Y. M. Bishop, S. E. Fienberg, and P. W. Holland. *Discrete Multivariate Analysis: Theory and Practice*. Springer, 2007.

[5] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD Conference*, pages 265–276, 1997.

[6] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, pages 3–18, 2009.

[7] 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):1134–1168, 2006.

[8] J. Cheng, Y. Ke, and W. Ng. delta-tolerance closed frequent itemsets. In *ICDM*, pages 139–148, 2006.

[9] S. Garg, K. Ramamritham, and S. Chakrabarti. Web-cam: Monitoring the dynamic web to respond to continual queries. In *SIGMOD Conference*, pages 927–928, 2004.

[10] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.

[11] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[12] A. Kashyap, V. Hristidis, and M. Petropoulos. Facetor: cost-driven exploration of faceted query results. In *CIKM*, pages 719–728, 2010.

[13] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210, 2006.

[14] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. Flexrecs: expressing and combining flexible recommendations. In *SIGMOD Conference*, pages 745–758, 2009.

[15] S. B. Roy, H. Wang, G. Das, U. Nambiar, and M. K. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *CIKM*, pages 13–22, 2008.

[16] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, pages 168–182, 1998.

[17] N. Sarkas, N. Bansal, G. Das, and N. Koudas. Measure-driven keyword-query expansion. *PVLDB*, 2(1):121–132, 2009.

[18] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *ICDT*, pages 89–103, 2010.

[19] K. Stefanidis, M. Drosou, and E. Pitoura. "you may also like" results in relational databases. In *PersDB*, 2009.

[20] L. Szathmary, A. Napoli, and P. Valtchev. Towards rare itemset mining. In *ICTAI (1)*, pages 305–312, 2007.

[21] N. Tintarev and J. Masthoff. Designing and evaluating explanations for recommender systems. In *Recommender Systems Handbook*, pages 479–510. 2011.

[22] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD Conference*, pages 15–26, 2010.

[23] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *SIGMOD Conference*, pages 535–548, 2009.

[24] K. Wang, Y. Jiang, and L. V. S. Lakshmanan. Mining unexpected rules by pushing user dynamics. In *KDD*, pages 246–255, 2003.