Generalized Précis Queries for Logical Database Subset Creation

Alkis Simitsis
Nat'l Techn. Univ. of Athens
Athens, Greece
asimi@dbnet.ece.ntua.gr

Georgia Koutrika
University of Athens
Athens, Greece
koutrika@di.uoa.gr

Yannis Ioannidis
University of Athens
Athens, Greece
yannis@di.uoa.gr

Abstract

As a large fraction of available information resides in databases, the need for facilitating access for the large majority of users becomes increasingly more important. Précis queries are free-form queries that generate entire multi-relation databases, which are logical subsets of existing ones. A logical subset contains not only items directly related to the given query selections but also items implicitly related to them in various ways with the purpose of providing to the user much greater insight into the original data. This paper is concerned with the definition and generation of logical database subsets based on précis queries under a generalized perspective that removes several restrictions of previous work and handles queries containing multiple terms combined using the operators AND, OR, and NOT.

1 Introduction

Emergence of the World Wide Web has made information access possible to a growing number of people. As libraries, museums, and other organizations publish their electronic contents on the Web, a large fraction of information resides in databases. The need for facilitating access to information stored in databases becomes increasingly more important. Towards this direction, current commercial and research efforts have adopted an Information-Retrieval approach and have focused on keyword-based searches over databases that relieve the user of the need to have any knowledge about schemas, query languages or even the schema of a particular database, to form their own structured queries [1, 2, 3, 4, 5, 6, 8].

Recently, précis queries were introduced as free-form queries that generate entire multi-relation databases, which are logical subsets of existing ones, instead of individual relations [7]. A logical subset contains not only items directly related to the given query selections but also items implicitly related to them in various ways. Its purpose is to provide to the user greater insight into the original data and it may be used for different purposes ranging from data extraction to information discovery. Thus, the logical database subset that should be extracted from a database given a query may vary depending on various factors such as the application, the type of the query, and the user issuing the query. Consequently, supporting précis queries using a relational database system is not straightforward.

This paper is concerned with the generation of logical database subsets based on précis queries. Earlier work has been restricted to précis queries with a single keyword, which was searched for in all attributes of all database relations. Although a step forward, this is still quite restrictive. For example, single-keyword précis queries on a museum database would allow one to find everything related to "Da Vinci", but not everything related to "Da Vinci" and "Michelangelo". In this paper, we examine logical database subsets and précis queries under a generalized model. In doing so, we address several technical challenges that arise on the way.

Contributions. In particular, the contributions of this paper are the following. We extend précis query semantics considering that queries may contain multiple terms combined using the operators AND, OR, and NOT. Based on these extended semantics, we consider the logical subset of a database, based not only on purely syntactic constraints, but also in terms of relevance to a given query. Moreover, we provide algorithms for the generation of logical database subsets. Finally, we present a set of experimental results that demonstrate the efficiency and benefits of our approach.

Due to space limitations, more details about this work can be found in the long version of this paper [9].

2 General Framework

A *Database Schema Graph* $\mathbf{G}(\mathbf{V}, \mathbf{E})$ is a directed graph corresponding to a database schema $\mathbf{D}$. There are two types of nodes in $\mathbf{V}$: (a) relation nodes, $\mathbf{R}$, one for each relation in the schema; and (b) attribute nodes, $\mathbf{A}$, one for each attribute of each relation in the schema. Likewise, edges in $\mathbf{E}$

1-4244-0803-2/07/$20.00 ©2007 IEEE

are: (a) projection edges, $\mathbf{\Pi}$, each one connects an attribute node with its container relation node, representing the possible projection of the attribute in the system's answer; and (b) join edges, $\mathbf{J}$, from a relation node to another relation node, representing a potential join between these relations. A weight $w \in [0, 1]$ assigned to an edge of a graph $\mathbf{G}$ represents the significance of the association between the nodes connected. Formally, a database schema graph is a directed graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$, where: $\mathbf{V} = \mathbf{R} \cup \mathbf{A}$ and $\mathbf{E} = \mathbf{\Pi} \cup \mathbf{J}$.

We consider queries formulated as combinations of terms with the use of logical operators. A term may be a word, e.g., "Leonardo", or a phrase, e.g., "Mona Lisa", enclosed in quotation marks. Operators include AND, OR, and NOT. Given a database D and a query Q, we define as *initial tuple* one in which at least one query term has been found, and as *initial relation* any database relation that contains at least one initial tuple.

A *logical database subset* L of D contains a set of relations that are a subset of those in the original database. For each relation in L, its set of attributes in L is a subset of its set of attributes in D and its set of tuples is a subset of the set of tuples in the original relation (when projected on the set of attributes that are present in the result). Similarly to the database schema graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$, we consider the logical subset schema graph $\mathbf{G}'(\mathbf{V}', \mathbf{E}')$ as a directed graph corresponding to a logical subset L.

Query Semantics. The result of applying Q on database D given a set of constraints C is a *logical database subset* L of D that satisfies the following:

- In the case of OR-semantics, L contains initial tuples for Q and any other tuple in D that is transitively reachable by *some* initial tuple through joins on $\mathbf{G}$, subject to the constraints in C.
- In the case of AND-semantics, L contains any tuple in D (including initial tuples) that is transitively reachable by *all* keywords combined with AND in Q, subject to the constraints in C.
- In the case of NOT-semantics, L contains any tuple in D (including initial tuples), *except* those that are transitively reachable by keywords associated with NOT in Q, subject to the constraints in C.

Example. *Consider a museum collection and the following queries issued.*

q_1: "$Da\ Vinci$" AND "$Painting$"
q_2: "$Da\ Vinci$" NOT "$Painting$"

The answer of q_1 would contain joining tuples in which all terms are found plus all tuples that are connected to these in various ways. Thus, the answer would contain information about paintings created by Da Vinci, paintings depicting Da Vinci, and so forth. The answer of q_2 would contain information about Da Vinci except anything related to paintings. Thus, all tuples of q_1 are excluded from the logical subset of q_2.

What prevents a query from returning the entire database as an answer is the set of constraints C that restricts the schema and tuples included in the logical database subset. Such constraints may be stored in the system or specified by the user at query time and their particular form depends on the application and the user characteristics. For example, given a large database, a developer who needs a smaller subset that conforms to the original schema for application testing, may provide *structural constraints* on the database schema graph itself, such as # $of\ relations$, # $of\ attributes\ per\ relation$, and # $of\ joins\ (per\ path)$. Alternatively, a web user who is interested in specific items, may provide *relevance constraints* on the weights on the database schema graph. Different sets of weights and/or different constraints on them result in different answers for the same précis query, offering adjustability and flexibility.

Schema of a Logical Subset. Given a database D and an AND/NOT-query Q, an *initial subgraph* (*IS*) is a rooted DAG $S_G(V_G, E_G)$ on the database schema graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ such that: (*a*) V_G contains *at least* one initial relation per query term, along with other relations that interconnect those, (*b*) E_G is a subset of E interconnecting the nodes in V_G, and (*c*) the root and *all* sink relations are initial relations. In an OR-query Q, there are at least as many initial subgraphs as terms in Q, each with an initial relation as its sole node.

Given a query Q, a database D, constraints C, and an initial subgraph S_G, an *expanded graph* is a connected subgraph on the database schema graph $\mathbf{G}$ that includes S_G and satisfies C. The set of all possible expanded subgraphs comprises the schema of the logical database subset $\mathbf{G}'$ that contains the most relevant information for the given query based on the constraints provided.

3 Logical Subset Schema Creation

Given a query Q, constraints C, and the database schema graph $\mathbf{G}$, initial relations are retrieved with the use of an inverted index. (No tuples are retrieved in this phase.) The creation of the LS schema is realized in two phases: the *initial subgraph creation* and the *initial subgraph expansion*.

The first phase performs the construction of the set $\mathbf{S_G}$ of initial subgraphs that correspond to Q. For *OR*-semantics, each initial relation comprises an initial subgraph in $\mathbf{S_G}$. The case of *AND*-semantics is confronted by the algorithm *FIS*. For each combination ξ of initial relations containing all terms in Q, the most significant initial subgraph subject to constraints C, if such subgraph exists, is placed in $\mathbf{S_G}$. For this aim, a best-first traversal of graph $\mathbf{G}$ is performed, starting from all initial relations. Hence, multiple subgraphs are progressively built. Each time an initial subgraph is identified that interconnects a different combination of the initial relations, containing all query terms, it is

Algorithm Find Initial Subgraphs (FIS)

Input: a database schema graph $\mathbf{G}(\mathbf{E}, \mathbf{V})$, constraints, and a set of initial relations $\mathbf{IR}$
Output: a set of initial subgraphs $\mathbf{S_G}$
Begin
0. $QP := \{\}$, $\mathbf{G'} := \mathbf{G}$
1. Foreach $R_i \in \mathbf{IR}$ {
   1.1 mark each relation R_i in $\mathbf{G'}$ with different $s\text{-}id$
   1.2 Foreach $e(R_i, x) \in \mathbf{E}$, $x \in \mathbf{V}$ {
      1.2.1 If e satisfies constraints {
         $w_{sid} := f_G(w_e)$
         mark respective e in $\mathbf{G'}$ with $s\text{-}id$
         add(QP, <e, $s\text{-}id$>)
   }}}
2. While QP not empty and constraints hold {
   2.1 get head <$e(R_i, R_j)$, $s\text{-}id$> from QP
   2.2 If destination relation R_j is not marked in $\mathbf{G'}$ and subgraph $s\text{-}id$ is retained acyclic {
      2.2.1 mark respective R_j in $\mathbf{G'}$ with $s\text{-}id$
      2.2.2 Foreach $e'(R_j, x) \in \mathbf{E}$, $x \in \mathbf{V}$ {
         If e' satisfies constraints {
            $w_{sid \cup e'} := f_G(W_G \cup w_{e'})$
            mark respective e' in $\mathbf{G'}$ with $s\text{-}id$
            add(QP, <e, $s\text{-}id$>)
      }}}
   2.3 If subgraph with $s\text{-}id$ contains new combination ξ {
      2.3.1 drop from subgraph all sink nodes n, s.t. $n \notin \xi$
      2.3.2 add subgraph in $\mathbf{S_G}$
   }}
3. return $\mathbf{S_G}$
End

Figure 1. Algorithm *FIS*

placed in $\mathbf{S_G}$. The logical operator *NOT* is confronted in the same way as *AND* in this phase. In a further step, the $\mathbf{S_G}$ produced is enriched with the appropriate attributes and projection edges w.r.t. constraints.

In the second phase, from each initial subgraph $S_G \in \mathbf{S_G}$, an expanded subgraph is derived with respect to C (algorithm *EIS*). *EIS* extends initial subgraphs by considering additional relations, in order to collect information "around" initial relations that is related to the query. Therefore, it builds a set of expanded subgraphs from $\mathbf{S_G}$, subject to the constraints, which comprises the LS schema graph $\mathbf{G'}$. New edges are gradually added in a subgraph $S_G \in \mathbf{S_G}$ in order of weight, as long as the target relation is significant for *all* initial relations in S_G with respect to C.

Theorem. Given a set of initial subgraphs and constraints C, algorithm *EIS* constructs correctly the set of expanded subgraphs, i.e., for each initial subgraph S_G, it finds *all* relations around S_G, such that each of them is significant for all initial relations in S_G with respect to constraints C.

Proof. The proof is omitted due to space considerations.

Algorithm Extend Initial Subgraphs (EIS)

Input: a database schema graph $\mathbf{G}(\mathbf{E}, \mathbf{V})$, constraints, a set of initial subgraphs $\mathbf{S_G}$, and a list QP
Output: LS schema graph $\mathbf{G'}(\mathbf{E'}, \mathbf{V'})$
Begin
1. While QP not empty and constraints hold {
   1.1. get head <$e(R_i, R_j)$, $s\text{-}id$> from QP
   1.2. If destination R_j is not marked on $\mathbf{G}$ (i.e. $R_j \notin V'$), is significant for S_G, and satisfies constraints {
      1.2.1 mark corresponding node on $\mathbf{G}$
      1.2.2 create a new node in $\mathbf{V'}$ for R_j
      1.2.3 create attribute nodes and projection edges in $\mathbf{G'}$ for attributes of R_j satisfying constraints
   }
   1.3. If $R_j \in \mathbf{V'}$, $e(R_i, R_j) \notin \mathbf{E'}$ and e satisfies constraints { 1.3.1 insert e in $\mathbf{E'}$ }
   1.4. annotate $e \in \mathbf{E'}$ with $s\text{-}id$
   1.5. update S_G identified by $s\text{-}id$
   1.6. Foreach join edge $e'(R_j, x) \in \mathbf{E}$, $x \in \mathbf{V}$, that retains S_G acyclic and satisfies constraints {
      1.6.1 add(QP, <e', $s\text{-}id$>)
   }}
2. return $\mathbf{G'}$
End

Figure 2. Algorithm *EIS*

4 Logical Subset Population

Given a database D and the schema graph $\mathbf{G'}$ of the logical database subset L for a query Q and constraints C, L contains the set of relations and attributes determined by the graph $\mathbf{G'}$ and a subset of tuples from D such that: $\forall t_j \in R_i, \forall R_i$ belonging to $\mathbf{G'}$, the following hold (based on the join edges on $\mathbf{G'}$): (*a*) t_j does not contain any query term associated with NOT in Q and does not transitively join to any initial tuple containing such query terms; (*b*) in case of OR-semantics, t_j is an initial tuple or transitively joins to an initial tuple; and (*c*) in case of AND-semantics, t_j transitively joins to initial tuples containing all query terms that are not contained in itself.

Naïve approach. In order to populate a logical subset, one approach is to consider the set of subgraphs marked on $\mathbf{G'}$, and for each one, build an appropriate query that retrieves tuples taking into account the initial relations contained in this subgraph and the semantics of query Q. The inverted index is used for retrieval of the id's of initial tuples. These queries are executed, and results obtained are used to populate each relation in the logical database subset L. Special care is required so that duplicate tuples and tuples not satisfying constraints are not inserted in the relations of the result. This approach is called *NaiveLSP*.

Algorithm PLSP. In [7], a different approach is used for the population of logical subsets corresponding to single-

term queries. The logical subset is generated by a series of simple selection queries without joins. In particular, initial tuples are retrieved first; then, tuples from any other relation in L are retrieved based on a list of values for the attribute that joins this relation to the logical subset. A heuristic is used in order to reduce the number of queries executed: if a relation in L collects tuples that transitively join to more than one initial relation, then the algorithm tries to collect them all, before joining another relation to this one.

We extend these ideas for the generalized précis queries under the query semantics considered in this work. The algorithm *PLSP* (Figure 3) consists of two phases. First, it populates initial subgraphs in order of decreasing weight (Ln: 1). Then, more tuples are retrieved and inserted into L by join queries starting from relations in the initial subgraphs and transitively expanding on $\mathbf{G}'$ (Ln: 2). For this purpose, a best-first traversal of the graph is performed. A critical observation is that subgraphs defined on $\mathbf{G}'$ may share joins and these may be executed multiple times. In order to minimize the number of joins executed and to avoid creating duplicate tuples, a join from R_i to R_j is not executed, w.r.t. the given constraints, until all subgraphs in which this join belongs have populated relation R_i.

Algorithm LSP. The algorithm *LS* Population, *LSP*, populates each expanded subgraph S of $\mathbf{G}'$ as follows. Initial relations in S that contain terms combined with AND are considered in increasing order of the estimated number of their tuples in the logical subset, by taking into account that: (*a*) a keyword in a relation may be found in more than one attribute, (*b*) more than one keyword may be found in a relation, and (*c*) the frequencies of keyword occurrences in a relation kept in the inverted index. Next, each time, the algorithm populates $\mathbf{G}'$ with the initial tuples of the smallest initial relation stored, along with all tuples that can be transitively joined with these initial tuples. Each tuple stored in L is marked; thus, duplicate tuples are not created. When a tuple that contains a NOT-term is found, its set of initial relations is emptied and this tuple and any tuple joining to it in the results obtained so far, will not be produced. Finally, when all initial relations of S have been visited, *LSP* examines the next expanded subgraph. For the interest of space, we omit the formal representation of *LSP*.

5 Experiments

We conducted experiments to evaluate the efficiency of the methods proposed taking into consideration the following parameters: (*a*) *the number of subgraphs #S* that comprise a logical database subset, which depends on the number of query terms and the data; (*b*) *the number of relations per subgraph #RS*, which is determined by the constraints provided; (*c*) *the number of initial relations per subgraph #IRS*, which depends on the number of query terms and the data (for OR-semantics, it is equal to 1); (*d*) *the number of tuples in the logical subset #TLS*, which also depends on the data; and (*e*) *the database size #DB*, which is considered as the number of relations in the whole database. Times shown in the results are in seconds.

Algorithm Progressive LS Population ($PLSP$)

Input: LS schema graph $\mathbf{G}'(\mathbf{E}', \mathbf{V}')$, constraints, and a set of initial subgraphs $\mathbf{S_G}$
Output: logical subset *L*
Begin
0. $QP := \{\}$
1. Foreach initial subgraph $S_G \in \mathbf{S_G}$ satisfying constraints {
   1.1 execute query corresponding to S_G
   1.2 Foreach relation R_j in S_G {
      1.2.1 populate relation R_j with result tuples
      1.2.2 annotate tuples in R_j with matching s-id's from $\mathbf{s\text{-}id's}$
      1.2.3 $\{QP, \mathbf{G}'\} \leftarrow$ addinQP(R_j,$\mathbf{G}'$,QP,constraints)
   }}
2. While ((QP not empty or $\exists$ joins in $\mathbf{G}'$ not fully executed) and (constraints hold)) {
   2.1 If QP is not empty {
      2.1.1 get head <$e(x, R_j)$, $\mathbf{s\text{-}id's}$> from QP, $x \in \mathbf{V}'$
      2.1.2 populate R_j with $ExeJoin$(e,$\mathbf{s\text{-}id's}$,constraints)
      2.1.3 annotate tuples in R_j with matching s-id's from $\mathbf{s\text{-}id's}$
      2.1.4 $\{QP, \mathbf{G}'\} \leftarrow$ addinQP(R_j,$\mathbf{G}'$,QP,constraints)
      Else
      2.1.5 populate R_j with $ExeJoin$(most important pending join e in $\mathbf{G}'$ with destination R_j, $\mathbf{s\text{-}id's}$,constraints)
      2.1.6 annotate tuples in R_j with matching s-id's from $\mathbf{s\text{-}id's}$
      2.1.7 $\{QP, \mathbf{G}'\} \leftarrow$ addinQP(R_j,$\mathbf{G}'$,QP,constraints)
   }}
3. Return *L* as the $\mathbf{G}'$ populated with tuples
End

addinQP(relation R_j, graph $\mathbf{G}'$, list QP, constraints) {
 Foreach join edge $e(R_j, x) \in \mathbf{E}'$, $x \in \mathbf{V}'$ {
  mark those s-id's of e that have already populated R_j
  If e has all s-id's marked or due to constraints
   { add(QP,<$e(R_j, x)$, $\mathbf{s\text{-}id's}$>) }
}}

Figure 3. Algorithm $PLSP$

LS *Schema Creation*. Figure 4a shows the behavior of the LS schema creation procedure, *LSSC*, for varying database size, ranging from 10 to 100. As defaults, we have used *#IRS*=4. Figure 4b depicts execution times of the algorithm for varying number of initial relations, ranging from 1 to 10, and for two databases comprising 20 and 30 relations respectively. In both cases, *LSSC* needs less than 0.7 sec to build initial subgraphs containing combinations of 1 to 10 initial relations.

LS *Population*. We compare the three methods presented

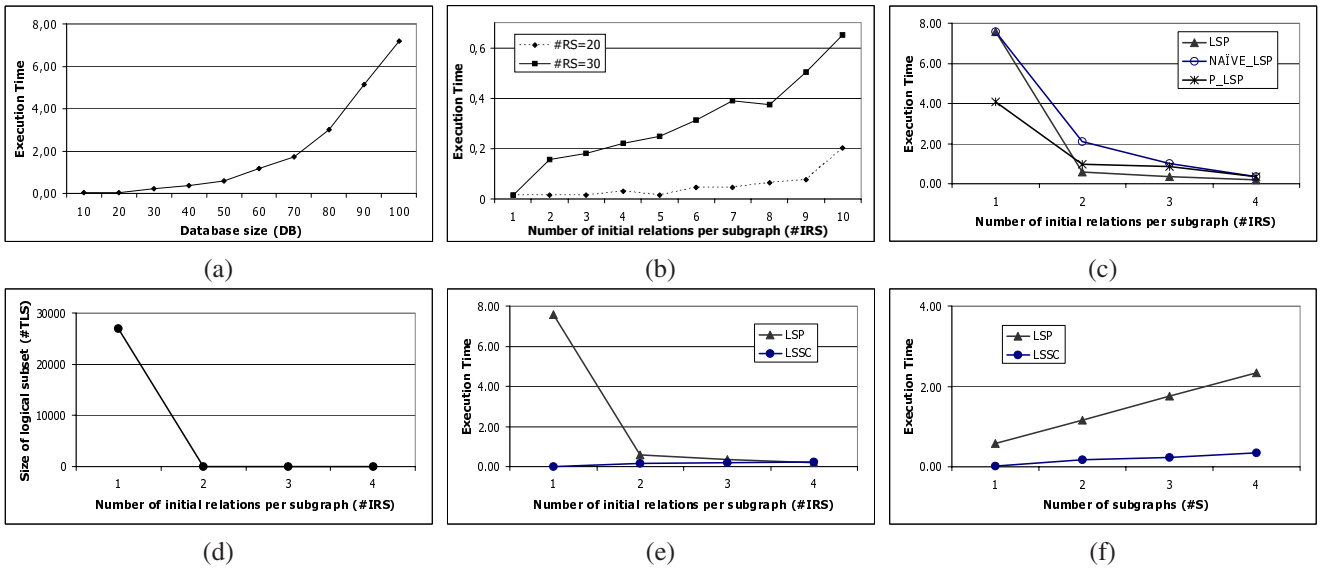

Figure 4. Experimental evaluation of logical subset characteristics

before: *NaiveLSP*, *PLSP*, and *LSP*. Figure 4c presents execution times for varying *#IRS*, *#S*=1, and *#RS*=4. *#IRS*=1 means that query terms are combined with OR, while *#IRS*>1 means that query terms are combined with AND. Overall, execution times decrease as the number of initial relations increases, because more initial relations result in a stricter query that generates a smaller answer. This is shown in Figure 4d: for *#IRS*>1, *#TLS* significantly decreases (< 100 tuples). Results not shown here due to space considerations show that performance of *NaiveLSP* and *PLSP* deteriorates significantly, whereas *LSP* remains more efficient for queries containing NOT.

Overall Performance. Finally, we present representative experiments regarding the overall performance of our approach. Figure 4e presents execution times for varying *#IRS*, *#S*=1, and *#RS*=4. Figure 4f compares execution times for *#S* ranging from 1 to 4, *#RS*=4, and *#IRS*=2.

The general observation is that although the LS schema creation, *LSSC*, and LS population, *LSP*, execution times depend on parameters, such as number of subgraphs and number of initial relations per subgraph, execution times of *LSP* determine the total time required for the construction of a logical database subset.

6 Conclusions

A logical subset of a database generated by a précis query contains not only items directly related to the query selections but also items implicitly related to them in various ways. In this paper, we elaborating on the idea of précis queries by allowing them to contain multiple keywords, combined with OR, AND, and NOT operators. We described algorithms that generalize the functionality and optimize the performance of précis queries, as the experiments conducted indicate.

An interesting approach for future work is the incorporation of the logical subset into the internals of the RDBMS and the possible extension of SQL language to support it.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[3] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into xml query processing. *Computer Networks*, 33(1-6), 2000.

[4] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[5] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[6] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on xml graphs. In *ICDE*, pages 367–378, 2003.

[7] G. Koutrika, A. Simitsis, and Y. Ioannidis. Précis: The essence of a query answer. In *ICDE*, 2006.

[8] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[9] A. Simitsis, G. Koutrika, and Y. Ioannidis. *Generalized Précis Queries for Logical Database Subset Creation*. Technical Report, 2006.