

CSE 736 Database Seminar

Mohan Kumar Padmanabhan

cse@buffalo

Papers Considered

- ✓ **Combining Keyword Search and Forms for Ad Hoc Querying of Databases**
 - Eric Chu, Akanksha Baid, Xiaoyong Chai, AnHai Doan, Jeffrey Naughton
 - Computer Sciences Department
 - University of Wisconsin-Madison
- ✓ **Keyword Searching and Browsing in Databases using BANKS**
 - Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, S. Sudarshan
 - Computer Science and Engineering Dept.
 - I.I.T. Bombay

Motivation

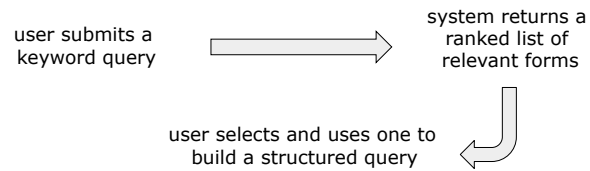
- General public is successful at using keyword search to discovering documents of interest in Internet search engines
- It is much more difficult to pose structured queries to satisfy information requests over structured databases
- Goal here is to explore techniques that assist users in posing ad hoc structured queries over relational databases

Google Example

The screenshot shows a Google search interface in Mozilla Firefox. The search query is "from new york to seattle". The results page shows several sponsored links and organic search results. A red box highlights a result for "Flights from New York, NY to Seattle/Tacoma, WA" with details for a flight from CheapTickets.com, including departure on 02/25 and return on 03/04. Other results include "Fly SEA to NY" from VirginAmerica.com, "Seattle to New York JFK" from jetblue.com, and "Flights from New York" from CheapOair.com.

Introduction

- It is easier to recognize a solution when presented with one
 - than constructing the solution from scratch
- Use keyword search to help the user find a manageably small set of relevant forms



Example

widom search

1. [related_people_AVG.jsp](#) (score : 1.0)
2. [related_people_GROUPSELECT.jsp](#) (score : 1.0)
3. [related_people_INTERSECT.jsp](#) (score : 1.0)
4. [related_people_Reg.jsp](#) (score : 1.0)
5. [related_people_SUM.jsp](#) (score : 1.0)
6. [related_people_UNION.jsp](#) (score : 1.0)
7. [co_author_AVG.jsp](#) (score : 0.9198291897773743)
8. [co_author_GROUPSELECT.jsp](#) (score : 0.9198291897773743)
9. [co_author_INTERSECT.jsp](#) (score : 0.9198291897773743)
10. [co_author_Reg.jsp](#) (score : 0.9198291897773743)
11. [co_author_SUM.jsp](#) (score : 0.9198291897773743)
12. [co_author_UNION.jsp](#) (score : 0.9198291897773743)
13. [related_people_CNT.jsp](#) (score : 0.8571430444717407)
14. [related_people_GROUPAVG.jsp](#) (score : 0.8571430444717407)
15. [related_people_GROUPCNT.jsp](#) (score : 0.8571430444717407)
16. [related_people_GROUPMAX.jsp](#) (score : 0.8571430444717407)
17. [related_people_GROUPMIN.jsp](#) (score : 0.8571430444717407)
18. [related_people_GROUPSUM.jsp](#) (score : 0.8571430444717407)
19. [related_people_MIN.jsp](#) (score : 0.8571430444717407)
20. [related_people_MAX.jsp](#) (score : 0.8571429252624512)
21. [co_author_CNT.jsp](#) (score : 0.788425087926772)

Example (cont'd)

widom search

- ☐ Which two people are related
- ☐ Which two people are co-authors
- ☐ Details about a Person
- ☐ Who gives a Tutorial in which conference
- ☐ Who gives a talk in which conference
- ☐ Who gives a talk in which Organization
- ☐ Who is related to which Organization
- ☐ Who is related to which Area/Topic
- ☐ Who has served in which conference
- ☐ Person who has given a Conference talk, Organization talk and Tutorial
- ☐ Who has authored which publication
- ☐ Details about a Publication

Example (cont'd)

Person

name op group =

homepage op organization =

title op country op

Options and Challenges

- How can one automatically generate a set of forms to support a wide range of queries?
- How specific or general should these forms be?
- How effective is keyword search in exploring this set of forms?
- What challenges arise in ranking the results of these keyword searches?

- Can users really use the result of a keyword search to identify forms useful in satisfying their information requests?

Dataset Considered

Entity tables: # rows	
person (id, name, homepage, title, group, organization, country)	68459
publication (id, name, booktitle, year, pages, cites, clink, link)	108972
topic (id, name)	736
organization (id, name)	163
conference (id, name)	170

Relationship tables:	
<i>Records two related persons and strength of this pair</i> related_people (rid, pid1, pid2, strength)	115436
<i>Records related person-topic pair and strength</i> related_topic (rid, pid, tid, strength)	114196
<i>Records related person-organization pair and strength</i> related_organization (rid, pid, oid, strength)	2436
<i>Records a person giving a tutorial in a conference</i> give_tutorial (rid, pid, cid)	132
<i>Records a person giving a talk in a conference</i> give_conf_talk (rid, pid, cid)	131
<i>Records a person giving a talk at an organization</i> give_org_talk (rid, pid, oid)	913
<i>Records a person serving in a conference and the assignment</i> serve_conf (rid, pid, cid, assignment)	3591
<i>Records a person as an author of a publication and the position of the person's name on the list of authors</i> write_pub (rid, pid, pub_id, position)	328410
<i>Records a pair of co-authors and strength</i> co_author (rid, pid1, pid2, strength)	56370

Approach

- Form generation
- Map keyword queries to forms
- Eliminate forms that do not produce answers with respect to a given keyword query
- Ranking and grouping forms
- Experiments and user study

Query Forms

Person

name *op* group =

homepage *op* organization =

title *op* country *op*

Query Forms

- When the form is empty, it maps to the template

```
SELECT *
FROM person
WHERE name op value AND homepage op value
AND title op value AND group op value AND
organization op value AND country op value
```

- A template with user-specified parameters corresponds to a SQL query

```
SELECT *
FROM person
WHERE organization = 'Microsoft Research'
```

UB CSE 736 Spring 2010

13

Form Generation

- Let D be a database instance and S_D be the schema of D
- Form generation:
 1. Specify a subset of SQL as the target language to implement the queries supported by forms
 2. Determine a set of "skeleton" templates specifying the main clauses and join conditions based on the chosen subset of SQL and S_D
 3. Finalize templates by modifying skeleton templates based on the desired form specificity
 4. Map each template to a form

UB CSE 736 Spring 2010

14

SQL'

Let $B =$ (**SELECT** *select-list*
FROM *from-list*
WHERE *qualification*
[**GROUP BY** *grouping-list*
HAVING *group-qualification*])

where

- **select-list** comprises a list of column names, and, if applicable, a list of terms having the form *aggop(column-name)*, with *aggop* being one of {MIN, MAX, COUNT, SUM and AVG}
- **from-list** is a list of tables
- **qualification** is a conjunction of the conditions of the form *expression op expression*. An expression is a column name or a constant, and *op* is one of the comparison operators {<, <=, =, >, >=, >, LIKE}
 - Note: we do not allow nested queries in FROM and WHERE clauses
- **grouping-list** and **group-qualification** are as defined in SQL-92 (i.e., no every or any in *group-qualification*)
- We consider queries of the form B [**UNION**|**INTERSECT** B]

UB CSE 736 Spring 2010

15

Skeleton Templates

- EX_{basic} :

```
SELECT *
FROM Ri
WHERE predicate-list
```
- EX_{FK} :

```
SELECT *
FROM give_tutorial t, person p, conference c
WHERE t.pid = p.id AND t.cid = c.id AND p.name
op expr AND ... AND c.name op expr
```
- EX_{EQ} :

```
SELECT non-key attributes from p
FROM give_tutorial t, give_conf_talk c,
give_org_talk o, person p
WHERE t.pid = c.pid AND c.pid = o.oid AND
o.pid = p.id AND p.name op expr AND ... AND
p.country op expr
```

UB CSE 736 Spring 2010

16

Form Specificity

- Fewer, more general forms
 - *Pro* - easier to find a form that supports the query a user has loosely in their mind
 - *Con* - the user may have difficulty in understanding and using this form, especially when he or she is not familiar with the data model and the query language
- Larger number of more specific forms
 - *Con* - harder to find a form that matches the user's specific information need
 - *Pro* - when one is found, the necessary customization to express the query is minor

UB CSE 736 Spring 2010

17

Form Specificity

- Form specificity
 - *Form complexity*, which refers to the number of parameters on a form
 - *Data specificity*, which refers to the number of parameters with fixed values on a form

The screenshot shows a web form titled "Person". It contains six input fields arranged in two columns. The first column has "name op", "homepage op", and "title op". The second column has "group =", "organization = Microsoft Research", and "country op". At the bottom right, there are two buttons: "Cancel" and "Submit".

UB CSE 736 Spring 2010

18

Form Specificity

- Map each skeleton template, which has only a SELECT-FROM-WHERE construct, to one large template supporting aggregation, GROUP BY and HAVING, and UNION and INTERSECT
- Such a multi-purpose query template could be too complex
- We reduce form complexity by dividing SQL' into subsets:
 - 1. SELECT:** the basic SELECT-FROM-WHERE construct
 - 2. AGGR:** SELECT with aggregation
 - 3. GROUP:** AGGR with GROUP BY and HAVING clauses
 - 4. UNION-INTERSECT:** a UNION or INTERSECT of two SELECT
- We do not consider data specific forms

UB CSE 736 Spring 2010

19

Mapping Query Templates to Forms

- To build a form for each query template, we use the following standard form components:
 - **Label:** for displaying text such as description for the form, the name of an attribute, a database constant, etc.
 - **Drop-down list:** for displaying a list of parameter values from which users can choose one. For example, we use a drop-down list to allow users to choose the target attribute for an aggregation.
 - **Input box:** for specifying a parameter value on the form
 - **Button:** for functions such as submit, cancel, and reset

UB CSE 736 Spring 2010

20

Automating Form Generation

- Template generator uses the aforementioned specification for SQL' and query classes
- Input: a data set and its schema
- A form designer can specify the desired form complexity and data specificity
- Output is a set of templates based on these configurations
- Scripts to transform these templates into forms and to add a form description to each form

Keyword Search for Forms

- Basic idea here is to treat a set of forms as a set of documents, then let users use keyword search to find relevant forms
- Form contains parameters, which are undefined until users fill out the form at query time
- **Naïve-AND** – user specifies a data value, we will get no answers
- **Naïve-OR** – some forms would be returned if the user includes in the query at least one *schema term*
 - Data terms would be ignored

Example

- Query: **Widom conference**
 - We like to know for which conferences a researcher named **Widom** has served on the program committee
- Assume **Widom** is a data term and **conference** is a schema term
- Using Naïve-AND, we would get no forms, since **Widom** does not appear on any forms
- Using Naïve-OR, we would ignore **Widom** and get all forms that contain **conference**

Keyword Search for Forms

- Data specific form – many combinations and high storage and maintenance costs
- Transform a user's keyword query by checking to see whether the terms from the query appear in the database
 - user-provided keyword appears both as a schema term and as a data term
 - keyword appears in multiple attributes, possibly of different tables
- Use **Double-Index OR (DI-OR)** and **Double-Index AND (DI-AND)**

Double-Index OR (DI-OR)

Input: A keyword query $Q = [q_1 q_2 \dots q_n]$

Output: A set of form-ids F'

Algorithm:

```
FormTerms = {}, F' = {}
// Replace any data terms with table names
for each  $q_i \in Q$ 
    if DataIndex( $q_i$ ) returns <table, tuple-id> pairs
        Add each table to FormTerms
    Add  $q_i$  to FormTerms //  $q_i$  could be a form term
// Get form-ids based on FormTerms
FormIndex(FormTerms) => F' // OR semantics
return F' // Ordered by ranking scores
```

DI-OR Example

- Query: **Widom conference**
- Using DI-OR, we would find that **Widom** appears in the **person** table
- The resulting rewritten keyword query would be **Widom person conference**, evaluated with OR semantics

DI-OR Summary

- Approach satisfies the new semantics
- Results are often too inclusive
- Approach similar to DI-OR but with AND semantics required
- Wrong to simply do one AND-query with all the terms in *FormTerms*
 - A data term may appear in multiple unrelated tables -> no form returned

Double-Index AND (DI-AND)

Input: A keyword query $Q = [q_1 q_2 \dots q_n]$

Output: A set of form-ids F'

Algorithm:

```
FormTerms = {}, F' = {}
// Replace any data terms with table names
for each  $q_i \in Q$ 
     $S_{q_i} = \{\}$  // Bucket for  $q_i$ 
    if DataIndex( $q_i$ ) returns <table, tuple-id> pairs
        for each table
            if table  $\notin$  FormTerms
                Add table to  $S_{q_i}$  and FormTerms
    if  $q_i \notin$  FormTerms
        Add  $q_i$  to  $S_{q_i}$  and FormTerms
```

Double-Index AND (DI-AND) (cont'd)

```
// Get form-ids based on  $Sq_i$ 
 $S_{Q'} = \text{EnumQueries}(\forall Sq_i)$  // Enumerate all
// unique queries, each having one
// term from each  $Sq_i$ 

for each  $Q' \in S_{Q'}$ 
   $\text{FormIndex}(Q') \Rightarrow F'$  // AND semantics on  $\text{FormIndex}$ 
return  $F'$  // Ordered by ranking scores
```

UB CSE 736 Spring 2010

29

DI-AND Example

- Query: **Widom conference**
- Using DI-AND, we would generate two queries:
 1. **person conference** and
 2. **Widom conference**
- Evaluate each with AND semantics, and return the union of the results
- In this case, **Widom conference** would lead to an empty result

UB CSE 736 Spring 2010

30

DI-AND Summary

- Large number of queries generated – but most of them are duplicates
- Query – mix of data terms
 - Add synonyms to a query based on a thesaurus during query evaluation
 - Add a set of synonyms to each form during form generation
- Selected and added a set of keywords to what we call a form profile for each form

UB CSE 736 Spring 2010

31

DI-AND Summary (cont'd)

- DI-AND can return forms that can never produce results with respect to the user query
 - When a search involves a table referenced by many other tables, DI-AND returns all the forms for all these tables, even though some may return no answer with respect to the user query
- We need to identify and filter these **dead forms** from the results

UB CSE 736 Spring 2010

32

Dead Forms Example

- Query: **John Doe**
- Assume **John Doe** appears in the **person** table, but is not involved in any relationship
 - That is, the **John Doe** tuple in **person** is not referenced by any tuple in any relationship table
- In addition to returning forms for the **person** table, DI-AND would return forms for all the relationship tables that reference **person**
- Since **John Doe** appears only in **person**, if the user enters **John Doe** in the **person.name** field on any of these join forms, they will return empty results

UB CSE 736 Spring 2010

33

Double-Index-Join

Input: A keyword query $Q = [q_1 q_2 \dots q_n]$

Output: A set of form-ids F'

Algorithm:

$FormTerms = \{\}, F' = \{\}, X = \{\}$

// Replace any data terms with table names

for each $q_i \in Q$

$Sq_i = \{\}$

if $DataIndex(q_i)$ returns $\langle table, tuple-id \rangle$ pairs

for each $table T$

let I be the set of $tuple-ids$ from T

if $T \notin FormTerms$

Add T to Sq_i and $FormTerms$

// New "join" step

$SchemaGraph(T)$ returns $refTables$

UB CSE 736 Spring 2010

34

Double-Index AND (DI-AND) (cont'd)

for each $refTable$

if $DataIndex(refTable:tid)$ is NULL for every $tid \in I$

$FormIndex(T AND refTable) \Rightarrow X$

if $q_i \notin FormTerms$

Add q_i to Sq_i and $FormTerms$

// Get form-ids based on form terms

$S_{Q'} = EnumQueries(\forall Sq_i)$

for each $Q' \in S_{Q'}$

$FormIndex(Q') \Rightarrow F'$

return $F' - X$ // Filter "dead" forms

UB CSE 736 Spring 2010

35

DISPLAYING RETURNED FORMS

UB CSE 736 Spring 2010

36

Ranking Forms

- The Lucene score for a query Q and a document D is:

$$\text{score}(Q,D) = \text{coord}(Q,D) * \text{queryNorm}(Q) * \sum_{t \in Q} [\text{tf}(t \text{ in } D) * \text{idf}(t)^2 * t.\text{getBoost}() * \text{norm}(t,D)]$$

Score factor based on # of query terms found in D

Normalizing factor

Term frequency of t in D

Inverse term frequency of t in D

Search time boost of t

Index time boost

UB CSE 736 Spring 2010

37

Lucene Scoring Terms

- The factors involved in Lucene's scoring algorithm are as follows:
 - tf** = term frequency in document = measure of how often a term appears in the document
 - idf** = inverse document frequency = measure of how often the term appears across the index
 - coord** = number of terms in the query that were found in the document
 - lengthNorm** = measure of the importance of a term according to the total number of terms in the field
 - queryNorm** = normalization factor so that queries can be compared
 - boost(index)** = boost of the field at index-time
 - boost(query)** = boost of the field at query-time

UB CSE 736 Spring 2010

38

Ranking Forms

- Very specific forms have problems
- Form specificity increases => number of forms created from each skeleton template increases
- Forms based on the same skeleton template (sister forms) become increasingly similar
- When a query is relatively vague, there is not enough information to determine the user's intent
- Many sister forms within each group => required form may get pushed low

UB CSE 736 Spring 2010

39

1. [related_people_AVG.jsp](#) (score : 1.0)
2. [related_people_GROUPSELECT.jsp](#) (score : 1.0)
3. [related_people_INTERSECT.jsp](#) (score : 1.0)
4. [related_people_Reg.jsp](#) (score : 1.0)
5. [related_people_SUM.jsp](#) (score : 1.0)
6. [related_people_UNION.jsp](#) (score : 1.0)
7. [co_author_AVG.jsp](#) (score : 0.9198291897773743)
8. [co_author_GROUPSELECT.jsp](#) (score : 0.9198291897773743)
9. [co_author_INTERSECT.jsp](#) (score : 0.9198291897773743)
10. [co_author_Reg.jsp](#) (score : 0.9198291897773743)
11. [co_author_SUM.jsp](#) (score : 0.9198291897773743)
12. [co_author_UNION.jsp](#) (score : 0.9198291897773743)
13. [related_people_CNT.jsp](#) (score : 0.8571430444717407)
14. [related_people_GROUPAVG.jsp](#) (score : 0.8571430444717407)
15. [related_people_GROUPCNT.jsp](#) (score : 0.8571430444717407)
16. [related_people_GROUPMAX.jsp](#) (score : 0.8571430444717407)
17. [related_people_GROUPMIN.jsp](#) (score : 0.8571430444717407)
18. [related_people_GROUPSUM.jsp](#) (score : 0.8571430444717407)
19. [related_people_MIN.jsp](#) (score : 0.8571430444717407)
20. [related_people_MAX.jsp](#) (score : 0.8571429252624512)
21. [co_author_CNT.jsp](#) (score : 0.788425087928772)
22. [co_author_GROUPAVG.jsp](#) (score : 0.788425087928772)
23. [co_author_GROUPCNT.jsp](#) (score : 0.788425087928772)

Grouping Forms

- Given a list of forms ordered by each form's score, our first approach comprises two steps
 - Form first-level groups by grouping consecutive sister forms with the same score.
 - In each first-level group, group forms by the four query classes described in [slide 15](#), and display the classes in the order of *SELECT*, *AGGR*, *GROUP*, and *UNION-INTERSECT*.

- Which two people are related
- Which two people are co-authors
- Which two people are related
- Which two people are co-authors
- Details about a Person
- Who gives a talk in which Organization
- Who is related to which Organization
- Who is related to which Area/Topic
- Who has served in which conference
 - Basic Form
- Aggregate Form(s)
- Group By Form(s)
- Union/Intersect Form(s)
- Who gives a talk in which Organization
- Who has authored which publication
- Details about a Publication

Grouping Forms

- When two sister forms have different ranking scores such that they are not consecutive, they join different first-level groups
- These groups still have the same description and could confuse users
- Solution: first group the returned forms by their table, then order the groups by the sum of their scores

- Which two people are related
- Which two people are co-authors
- Details about a Person
- Who gives a Tutorial in which conference
- Who gives a talk in which conference
- Who gives a talk in which Organization
- Who is related to which Organization
- Who is related to which Area/Topic
- Who has served in which conference
- Person who has given a Conference talk, Organization talk and Tutorial
- Who has authored which publication
- Details about a Publication

EXPERIMENTS

UB CSE 736 Spring 2010

45

Experimental Setup

- Search interface implemented with Perl CGI scripts
- MySQL as the back-end database
- Apache Web Server to host the service
- Forms
 - 14 Skeleton templates – one for each of the table
 - Based on query classes in [slide 15](#), 1 SELECT template, 5 AGGR templates(one for each aggregate), 6 GROUP templates (one for each aggregate and one without aggregate) and 2 UNION-INTERSECT templates
 - Totally $14 * 14 = 196$ forms

UB CSE 736 Spring 2010

46

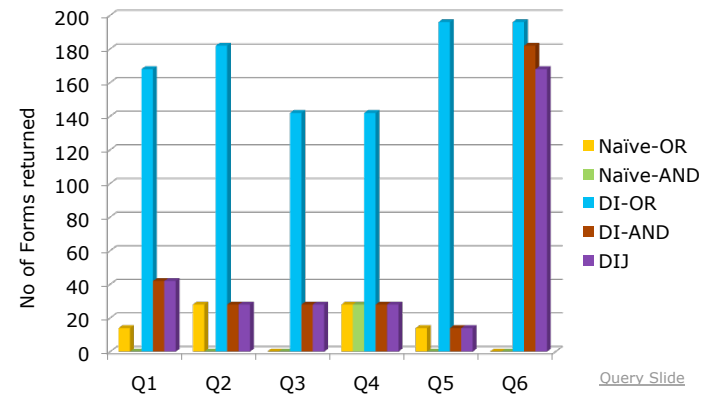
Queries Presented

- T1: Find all people who have given a tutorial at VLDB
 - "tutorial vldb"
- T2: Find topics of areas related to Jeff Naughton.
 - "jeff naughton research area"
- T3: Find people who have served as the SIGMOD PC chair
 - "sigmod chair"
- T4: Find the first author of all papers cited more than 5 times.
 - "paper citation"
- T5: Find the number of people who have co-authored a paper with David Dewitt.
 - "david dewitt coauthor"
- T6: Find people who have published with David DeWitt or Jeff Naughton.
 - "dewitt naughton"

UB CSE 736 Spring 2010

47

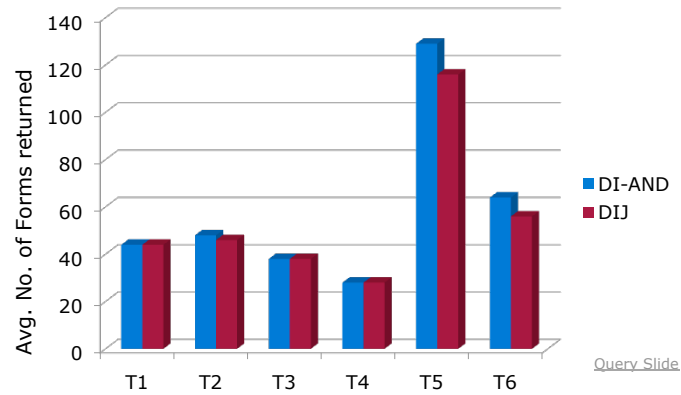
Results



UB CSE 736 Spring 2010

48

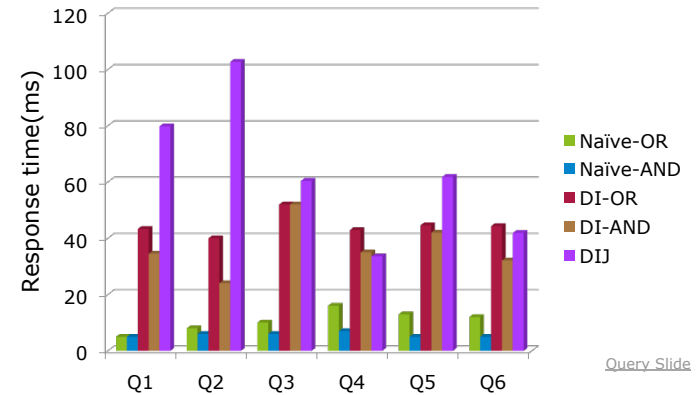
Results



UB CSE 736 Spring 2010

49

Results



UB CSE 736 Spring 2010

50

Ranking and Displaying Forms

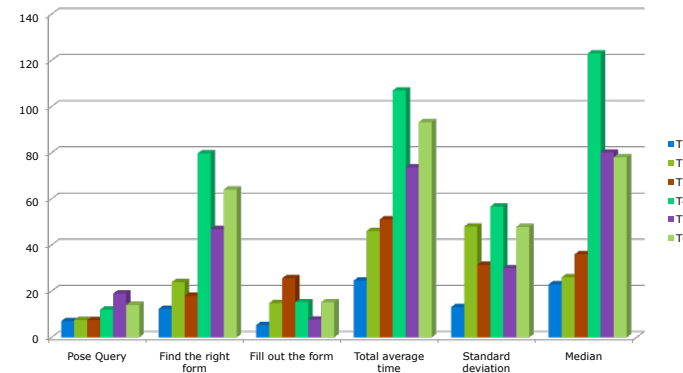
F1	Flat Rank				Group Rank			
	H	M	L	#F	H	M	L	#G
T1	1	1	1	44	1	1	1	3.14
T2	1	1	69	46	1	1	7	3.7
T3	1	1	1	38	1	1	1	2.7
T4	1	15	15	28	1	2	2	2
T5	4	21	21	116	1	2	4	11.57
T6	1	12	12	56	1	1	6	4

The highest (H), median (M), and the lowest (L) flat and group ranks for each queries, and the average number of forms (#F) and groups (#G) returned, based on the results of 7 users.

UB CSE 736 Spring 2010

51

User Interaction with Keyword Search and Forms



The breakdown of the time of using DIJ by 7 users

UB CSE 736 Spring 2010

52

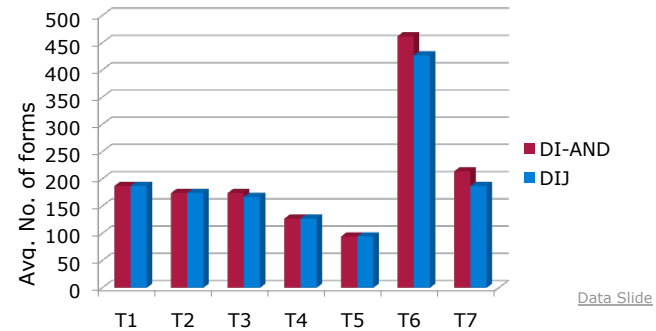
Impact of Adding Forms

- Forms for all combinations of equijoins involving 2 relationship tables and person table
- T7: Find people who have given a conference talk and given a tutorial.
 - "conference tutorial"

UB CSE 736 Spring 2010

53

Impact of Adding Forms - Results

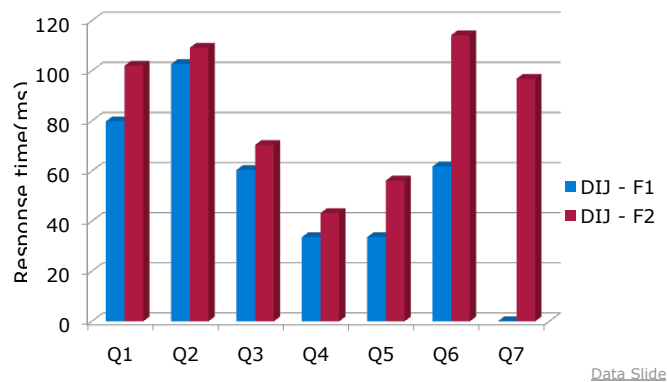


Data Slide

UB CSE 736 Spring 2010

54

Impact of Adding Forms - Results



Data Slide

UB CSE 736 Spring 2010

55

Related Work

- Query By Example
 - Skeleton tables presented to users
 - Users fill blanks in tables to specify constraints
 - Still require an understanding of relational model
- Basic keyword search over databases
 - Basic query specifications cannot be done
- Auto distinguish between schema and data terms
 - Little support for structured queries

UB CSE 736 Spring 2010

56

Issues Addressed

- Designing and generating forms in a systematic fashion
- Handling keyword queries that are a mix of data terms and schema terms
- Filtering out forms that would produce no results with respect to a user's query
- Ranking and displaying forms in a way that help users find useful forms more quickly

Scope of Future Work

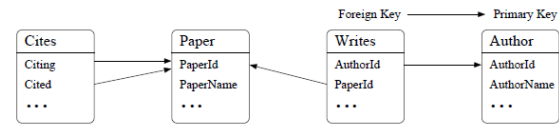
- Developing automated techniques for generating better form descriptions
- Exploring the tradeoffs between keyword search directly over the relational database and the above explained approach

Keyword Searching and Browsing in Databases using BANKS

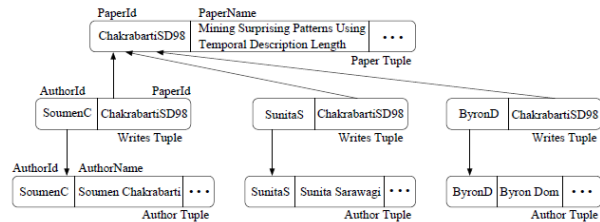
What is BANKS

- Browsing AND Keyword Searching
- Framework for keyword querying of relational databases.
- It makes joins implicit and transparent, and incorporates notions of proximity and prestige when ranking answers
- Novel, efficient heuristic algorithms for executing keyword queries

Dataset and Representation



(A) The Schema



(B) A Fragment of the Database

UB CSE 736 Spring 2010

61

BANKS Model

- Database modeled as directed graphs
 - Tuple being a node in the graph
 - Foreign-key-primary-key acting as directed edge
- Weights are assigned to the nodes and edges
- Nodes are identified corresponding to the search terms
- Answer to a query is a rooted directed tree
- Nodes fetched and ordered by a particular relevance score
- A heuristic backward expanding search algorithm used for computing query results

UB CSE 736 Spring 2010

62

Backward Expanding Search Algorithm

- For each keyword, set of nodes are identified which are relevant to the keyword
- For each node, a copy of Dijkstra's single source shortest path algorithm is executed
- Each copy runs backward to run a common vertex from which a forward path exists to at least one node in each set
- Such paths define a rooted directed tree with the common vertex as the root and the corresponding keyword nodes as the leaves
- The connection trees generated by the algorithm are only approximately sorted in the increasing order of their weights.

UB CSE 736 Spring 2010

63

Browsing BANKS

- Every displayed foreign key attribute value becomes a hyperlink to the referenced tuple
- Since the entire database is like a complex graph, various functionalities are provided
 - Projecting away columns
 - Selection on a column
 - Joining with foreign keys
 - Grouping by column
 - Sorting by a column

UB CSE 736 Spring 2010

64

Example Result

Table = PAPER

PAPERID	TITLE	YEAR
ChakrabartiSD98	Mining Surprising Patterns Using Temporal Description Length.	

Table = WRITES

NAME	PAPERID
Soumen Chakrabarti	ChakrabartiSD98

Table = AUTHOR

NAME	URL
Soumen Chakrabarti	

Table = WRITES

NAME	PAPERID
Sunita Sarawagi	ChakrabartiSD98

Table = AUTHOR

NAME	URL
Sunita Sarawagi	

UB CSE 736 Spring 2010

65

Browsing BANKS - Example

[STUDENTS, THESIS]		
SNAME	FEMAIL	TITLE
Nand Kumar Singh	sudhakar@aero.iitb.ac.in	Get column info Drop column Sort in Ascending order : of Sort in Descending order and Group by Group by prefix Join (FACULTY) Select
N. Shama Rao	mujumdar@aero.iitb.ernet.in	THROUGH THICKNESS ELASTIC CONSTANTS AND STRENGTHS OF ADVANCED FIBRE COMPOSITES
Mini N Balu	svs@math.iitb.ernet.in	Some Preservation Results in Mathematical Theory of Reliability

UB CSE 736 Spring 2010

66

Comparison: BANKS vs. Keyword-Forms

- In BANKS, the schema of tables are provided as hyperlinks. Browsing data is enabled by clicking these hyperlinks
- In Keyword-forms, schema is represented as forms and required data is entered in forms

UB CSE 736 Spring 2010

67

Comparison: BANKS vs. Keyword-Forms

- In BANKS, grouping of data done as part of the schema hyperlink while browsing the data
- In Keyword-forms, aggregate operations are done through forms. Appropriate forms need to be selected to get aggregated results

UB CSE 736 Spring 2010

68

Comparison: BANKS vs. Keyword-Forms

- Users need to know the schema in BANKS or the system needs to be able to map user-specified attributes to system attributes.
- In Keyword-Forms, schema elements are present in forms and no operators required in keyword search.

THANK YOU