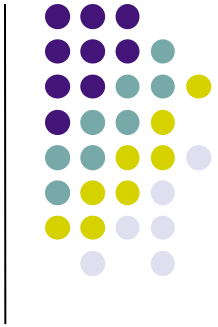


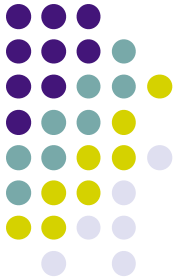
# Keyword Search on Form Results

Aditya Ramesh (Stanford)*
S Sudarshan (IIT Bombay)
Purva Joshi (IIT Bombay)

*Work done at IIT Bombay

VLDB 2011

# Keyword Search on Structured Data



- Allows queries to be specified without any knowledge of schema
- Lots of papers over the past 13 years
  - Tree as answers, Entities/virtual documents as answers, ranking, efficient search
- But why has adoption in the real world remained elusive?
  - Answers are not an a human usable form
  - Users forced to navigate through schema in the answers

**Rank:** 1 **Score:** 0.29653063 (es=0.33333334 , ns=0.185709)

- Table: role

*personid*=53205, *movieid*=42715, *character*=Cameo appearance (steerage dancer),

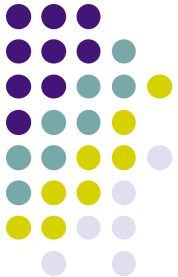
  - Table: person

  *id*=53205, *name*=**Cameron** James, *sex*=M,

  - Table: movie

  *id*=42715, *title*=**Titanic** (1997), *year*=1997, *rating*=7,

# Search on Enterprise Web Applications



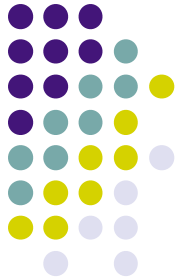
- Users interact with data through applications
  - Applications hide complexities of underlying schema
  - And present information in a human friendly fashion
- Applications have large numbers of forms
  - Hard for users to find information, built in search often incomplete
  - Forms sometimes map information only in one direction
    - e.g. student ID to name, but not from name to student ID
- Nice talk motivating keyword search on enterprise Web applications by Duda et al, CIDR 2007

http://univ.edu/acadrecords/studentinfo?ID=12345678

| | | | |
|---|---|---|---|
| **ID** | 12345678 | **Name** | Bill Gates |
| **Department** | Computer Science & Engineering | **Program** | Bachelor of Science |

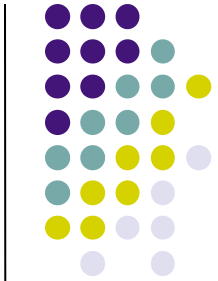
… grade, contact, and other information …

# Problem Statement



- System Model:
  - Set of forms, each taking 0 or more parameters
  - Result of a form = union of results of one or more parameterized queries
    - E.g. studentinfo form with parameter $ID
      - displays name and grades of the student
      1. select ID, name from student where ID = $ID
      2. select * from grades where ID = $ID
- Keyword search on form results
  - given set of keywords, return (form ID, parameter) combinations whose result contains given keywords
  - Ranked in a meaningful order

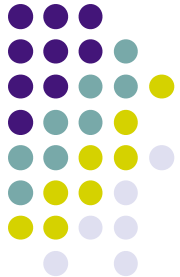
# Related Work



- Lots of papers on search (BANKS, Discover, DBXplorer, …)
  - Don't address presentation of results
- Precis, Qunits, Object summaries
  - Improve on presentation of information related to entities
  - But don't address search
- Predicate-based indexing (Duda et al. [CIDR 2007])
  - Materializes and indexes form results for all possible parameter values
  - But materialized results must be maintained
    - Same problem with virtual documents (Su and Widom [IDEAS05])
    - Efficient maintenance not discussed in prior work
    - Our experimental results show high cost even with efficient incremental view maintenance
- Find potentially relevant forms from a pre-generated set of forms

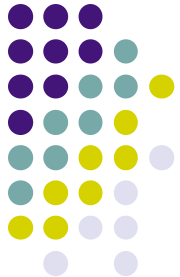
  Chu et al. (SIGMOD 2009, VLDB 2010)
  - But do not generate parameter values

# Assumptions and Safety



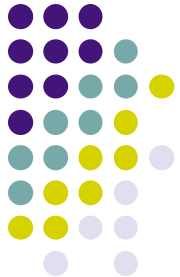
- *Form queries* take parameters which come directly from form parameters
  - Only mandatory parameters, no optional parameters
  - Parameters prefixed with $: e.g. $Id, $dept
  - E.g. $\Pi_{name}\sigma_{dept = \$dept}(prof)$
- Query Q: maps parameters P to results
- Inverted query IQ: maps keywords K to parameters P, s.t. Q(P) contains K
- Safety: inverted query may have infinite # of results
  - Q: $\Pi_{name}\sigma_{dept > \$dept}(prof)$
  - Q: $\Pi_{name}\sigma_{dept = \$dept \vee Id=\$Id}(prof)$

# Sufficient Conditions for Safety



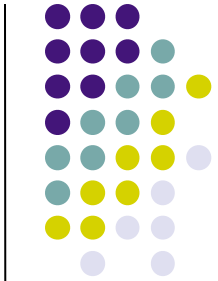
- Restrictions on form queries to ensure safety
  - Each parameter must be equated to some attribute
    - E.g. r.aj = $Pi;  r.aj is a called a parameter attribute
    - Above must appear as a conjunct in overall selection predicate
      - See paper for a few more restrictions for outerjoins and NOT IN/NOT Exists subqueries (antijoins)
- In some cases queries can be rewritten to satisfy above conditions
  - E.g. if parameter values for $P must appear in R(A), rewrite Q to  Q  $\sigma_{A=\$P}(R)$
- We handle some unsafe cases by using a "*" answer representation
  - e.g. (Form 1, $dept = 'CS' and $Id = *)

# Sufficient Conditions for Safety



- Restrictions on form queries to ensure safety
  - Each parameter must be equated to some attribute
    - E.g. r.aj = $Pi;  r.aj is a called a parameter attribute
    - Above must appear as a conjunct in overall selection predicate
      - See paper for a few more restrictions for outerjoins and NOT IN/NOT Exists subqueries (antijoins)
- In some cases queries can be rewritten to satisfy above conditions
  - E.g. if parameter values for $P must appear in R(A), rewrite Q to  $Q \bowtie \sigma_{A=\$P}(R)$
- We handle some unsafe cases by using a "*" answer representation
  - e.g. (Form 1, $dept = 'CS' and $Id = *)

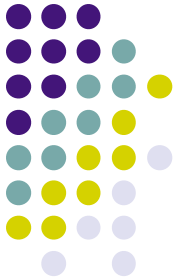
# Query Inversion 1:[1]



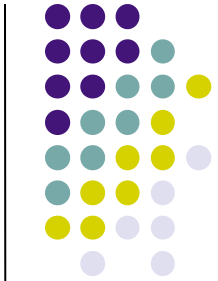
- Keyword Independent Inverted Query (KIIQ)
  - Intuition: Output parameter value along with result
    - for all possible parameter values
  - How?: Drop parameter predicate, e.g. *Id* = *$Id* and add parameter attribute, e.g. *Id*, to projection list
- Example:
  - Q= $\pi_{name}\,\sigma_{Id=\$Id}(prof)$ KIIQ= $\pi_{name,\,Id}(prof)$
- Issue: what if intermediate operation blocks parameter attribute from reaching top of query?
  - Selection/join: not an issue
  - Projection: Just add parameter attribute to projection list
  - Aggregation, etc: will see later.

[1] Acknowledgement: Idea of inversion arose during discussions with Surajit Chaudhuri

# Query Inversion 2:

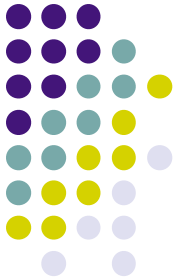


# Query Inversion 2:



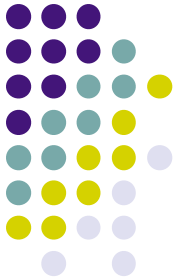
- Keyword Dependent Inverted Query (IQ)
- Add selection on keyword, and output only parameter values

# Query Inversion 2:



- Keyword Dependent Inverted Query (IQ)
- Add selection on keyword, and output only parameter values
  - IQ= $\pi_{\$params}(\sigma_{keyword\text{-}sels}(\mathrm{KIIQ}))$
  - E.g.: Q= $\pi_{name}\,\sigma_{Id=\$Id}(prof)$ Keyword query= {'John'}

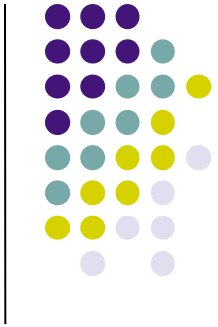
# Query Inversion 2:



- Keyword Dependent Inverted Query (IQ)
- Add selection on keyword, and output only parameter values
  - IQ= $\pi_{\$params}(\sigma_{keyword\text{-}sels}(\text{KIIQ}))$
  - E.g.: Q= $\pi_{name}\,\sigma_{Id=\$Id}\,(prof)$ Keyword query= {'John'}
  - KIIQ= $\pi_{Id}\,(prof)$

    IQ= $\pi_{Id}\,(\sigma_{Contains((name,\ Id),\ \text{“John”})}(prof))$
    - *Contains*((*R.A1,R.A2*,..),'K') efficiently supported using text indices

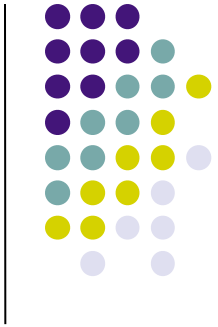
# Query Inversion 2:



- Keyword Dependent Inverted Query (IQ)
- Add selection on keyword, and output only parameter values
  - IQ= $\pi_{\$params}(\sigma_{keyword\text{-}sels}(\text{KIIQ}))$
  - E.g.: Q= $\pi_{name}\,\sigma_{Id=\$Id}(prof)$ Keyword query= {'John'}
  - KIIQ= $\pi_{Id}(prof)$

    IQ= $\pi_{Id}(\sigma_{Contains((name,\ Id),\ \text{“John”})}(prof))$
    - *Contains*((*R.A1,R.A2*,..),'K') efficiently supported using text indices
    - Parameter attributes like "Id" included in *Contains* even though if not in projection list,
- Multiple keywords: use intersection
  - E.g. K = {'John', 'Smith'}
  - $\pi_{Id}(\sigma_{Contains((name,\ Id),\ \text{“John”})}(prof))$
    $\cap\ \pi_{Id}(\sigma_{Contains((name,\ Id),\ \text{“Smith”})}(prof))$

# Queries With Multiple Relations



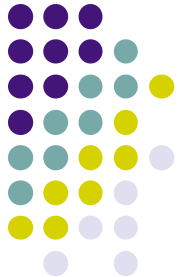
- Q= $\pi_{name,\ teaches.ctitle}\sigma_{\theta \wedge Id=\$Id}(prof \bowtie teaches)$
  - Id and Name attributes of *prof*
- KIIQ= $\pi_{Id,name,\ teaches.ctitle}\sigma_{\theta}(prof \bowtie teaches)$

- IQ= $\pi_{Id}\sigma_{Contains((Id,name,teaches.ctitle),\ 'John')}(\sigma_{\theta}(prof \bowtie teaches))$
- BUT most databases won't support keyword indexes across multiple relations, so we split into
  - $\pi_{Id}(\sigma_{Contains((Id,name),\ 'John') \vee Contains((teaches.ctitle),\ 'John')}(\sigma_{\theta}(prof \bowtie teaches)))$
  - Alternative using union more efficient in practice
    - $\pi_{Id}(\sigma_{Contains((Id,name),\ 'John')}(\sigma_{\theta}(prof \bowtie teaches)))$

      U $\pi_{Id}(\sigma_{Contains((teaches.ctitle),\ 'John')}(\sigma_{\theta}(prof \bowtie teaches)))$

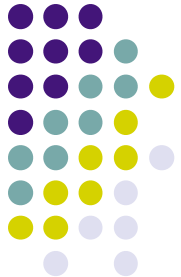
Note: *Contains* predicate will usually get pushed below join by query optimizer

# Complex Queries



- We focus on creating KIIQ
  - Key intuition: pull parameter attributes to top after removing parameter selection
  - Usual way of converting KIIQ to IQ
- Pulling Parameter Attribute above Aggregation
  - E.g. Q= $_{A}\gamma_{sum(B)}\,(\sigma_{\theta \wedge Id=\$Id}\,(E))$
  - KIIQ(Q) = $_{A,Id}\gamma_{sum(B)}\,(\sigma_{\theta}\,(E))$
- Intersection
  - Q= Q1 ∩ Q2
  - KIIQ(Q) = KIIQ(Q1)  KIIQ(Q2)
    - Note that parameters may be different for Q1 and Q2

# Complex Queries



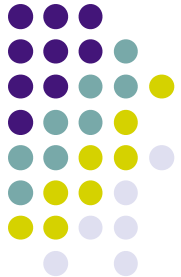
- We focus on creating KIIQ
  - Key intuition: pull parameter attributes to top after removing parameter selection
  - Usual way of converting KIIQ to IQ
- Pulling Parameter Attribute above Aggregation
  - E.g. Q= $_{A}\gamma_{sum(B)}\,(\sigma_{\theta \wedge Id=\$Id}\,(E))$
  - KIIQ(Q) = $_{A,Id}\gamma_{sum(B)}\,(\sigma_{\theta}\,(E))$
- Intersection
  - Q= Q1 ∩ Q2
  - KIIQ(Q) = KIIQ(Q1) ⋈ KIIQ(Q2)
    - Note that parameters may be different for Q1 and Q2

# Union Queries and Multiple Query Forms

- Forms with multiple queries
  - Form result = union of query results
  - Case of union queries is similar
  - E.g. Given Id as parameter, print name of professor and titles of courses taught
    - $\pi_{name}\sigma_{Id=\$Id}(prof)$ and $\pi_{ctitle}\sigma_{Id=\$Id}(teaches)$
- Case 1: Single keyword, same parameters for all queries
  - IQ = union of IQ for each query
  - E.g. $\pi_{Id}\sigma_{Contains((Id,name),\ 'John')}(prof)$

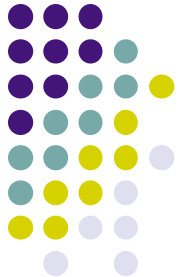
    U $\pi_{Id}\sigma_{Contains((Id,ctitle),\ 'John')}(teaches)$
- Does not work if different sets of parameters

# Multiple Query: Case 2



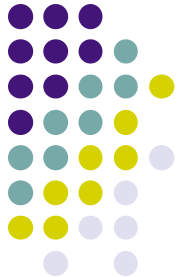
- Single keyword, different parameters across queries
  - E.g. $\pi_{name}\sigma_{Id=\$Id}(prof)$ and $\pi_{ctitle}\sigma_{dept=\$dept}(teaches)$
  - Define don't care value : '*' (matches all values)
    - $\pi_{Id,*}\sigma_{Contains((Id,name),\ 'John')}(prof)$
      $\cup\ \pi_{*,dept}\sigma_{Contains((dept,ctitle),\ 'John')}(teaches)$
- Multiple keyword, different parameters
  - Do as above for each keyword: $IQ_{k1}$, $IQ_{k2}$
  - Intersect results: $IQ_{k1} \cap IQ_{k2}$
  - Intersection not trivial due to '*'
  - Two approaches: KAT and QAT

# KAT: Keyword at a Time



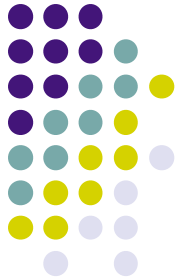
- Given queries Qi, Keywords Kj, and parameters Pk
  - For each Qi, Kj,
    - let QiKj = result of inverted query for Qi on Kj, with * for each parameter Pk not in Qi
  - Eg: Q1Kj: Id,Dept,*  Q2Kj: Id, *, Year
- Then combine answers, but using binding patterns
  - Using joins on non-* parameters
    - Q1K1-Q1K2: Join on Id, Dept
    - Q1K1-Q2K2, Q1K2-Q2K1: Join on Id
    - Q2K1-Q2K2: Join on Id, Year
- Details of optimizations and implementation in paper

# QAT: Query at a Time



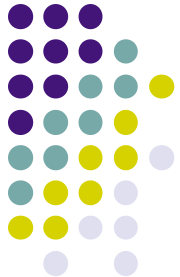
- Given queries Qi, and Keywords Kj
  - Create result QiKj for each keyword/query combo.
  - For each Qi combine results for all Kj, using bitmap
    - E.g. R1: (Id, Dept, bitmap), Bitmap: 1 bit per keyword
      R2:(Id, Year, bitmap)
- Then combine answers, but using binding patterns
  - Case 1: 2 queries: R = R1 ⟗ R2, and merge bitmaps
  - Case 2: All queries have same parameters
    - Again use full outerjoin and merge bitmaps
  - General case: R = R1 $U^+$ R2 $U^+$ R1 ⋈ R2
    - $U^+$ denotes outer union; merge bitmaps as before
- Finally, filter out results using bitmap
- Details in paper

# Other Cases



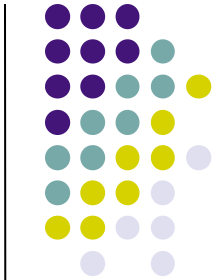
- Subqueries:
  - Trivial if subqueries don't have parameters
  - IN/EXISTS/SOME subqueries
    - Basic approach: decorrelate subqueries where possible
  - NOT IN, NOT EXISTS, ALL subqueries (antijoin)
    - disallow parameters in such subqueries (not safe)
- Static/application generated text in forms
  - Remove from keyword query if present in form

# Ranking



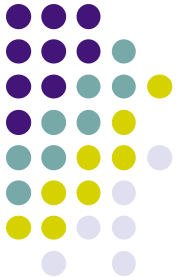
- Motivation for ranking
  - Form 1: Courses taught by particular instructor
  - Form 2: Courses in a particular department
    - Form result size much larger
  - Form 3: Courses taken by particular student
    - Form result is small, but many parameter values
- We rank forms, and rank parameters within forms
  - Ranking of forms
    - **Avg**: Average size of form result (precomputed)
    - **AvgMult**: Avg form result size * Number of distinct result parameter values
  - Ranking of parameters within form based on heuristics
    - E.g. current user ID/year/semester, department of current user
    - Special case for multiquery forms where keywords present in form prefix for some parameter value

# Performance Study



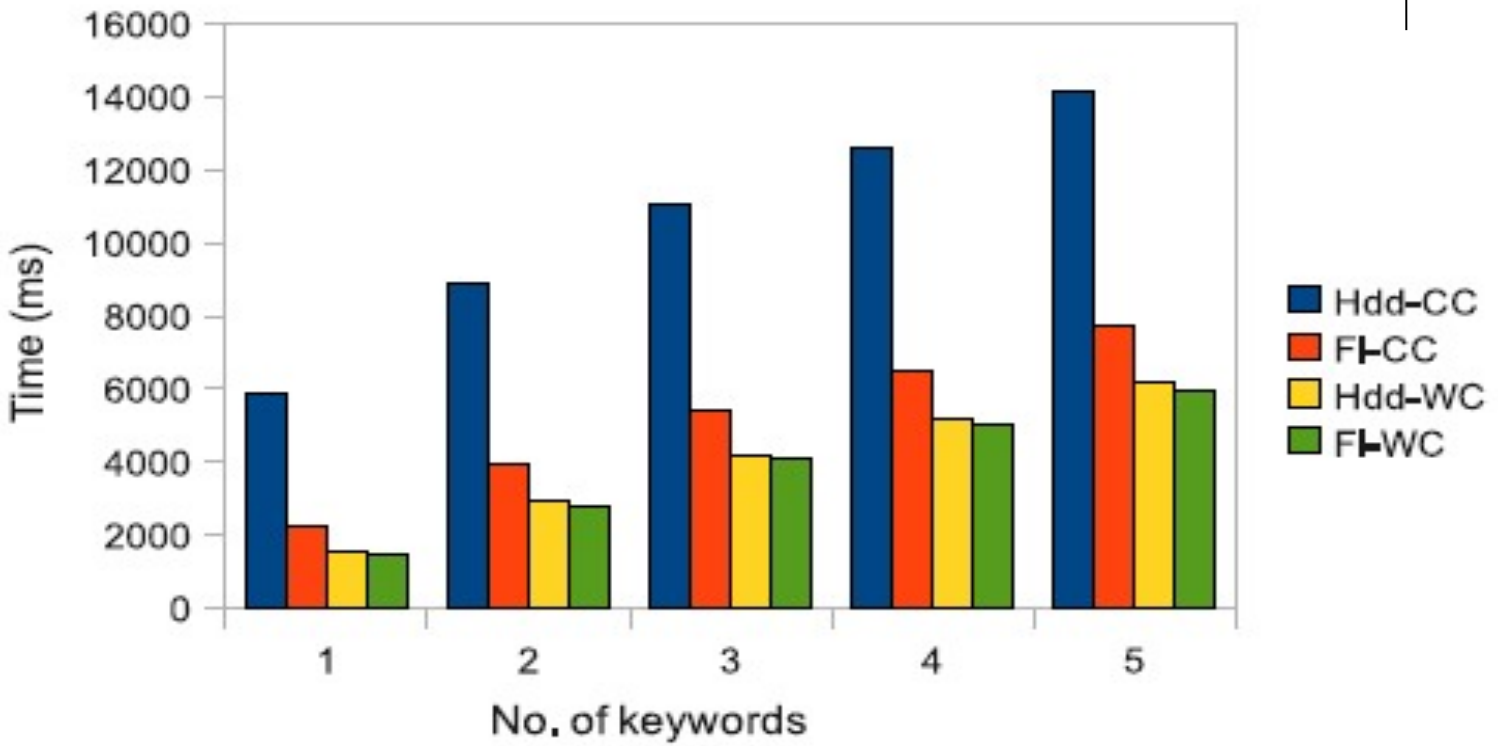
- IIT-Bombay Database Application
  - Real application
  - 90 forms,1 GB of data
- Queries used: model realistic goals for students and faculty
- Basic desktop machine with low end disk and generic 64 GB SATA MLC Flash disk

# Result/Ranking Quality



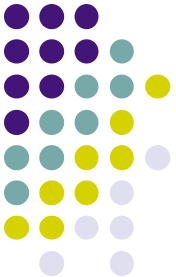
- Formulated several queries seeking information from academic database
- Found position of form returning desired answer
  - Average position:
    - 2.42 for AVG, 1.83 for AVGMULT
  - Max position: 6 for AVG, 3 for AVGMULT
- Heuristics for ranking parameters within form worked well
  - Need to generalize heuristics: future work

# Scalability with #Keywords + Hard Disk vs Flash

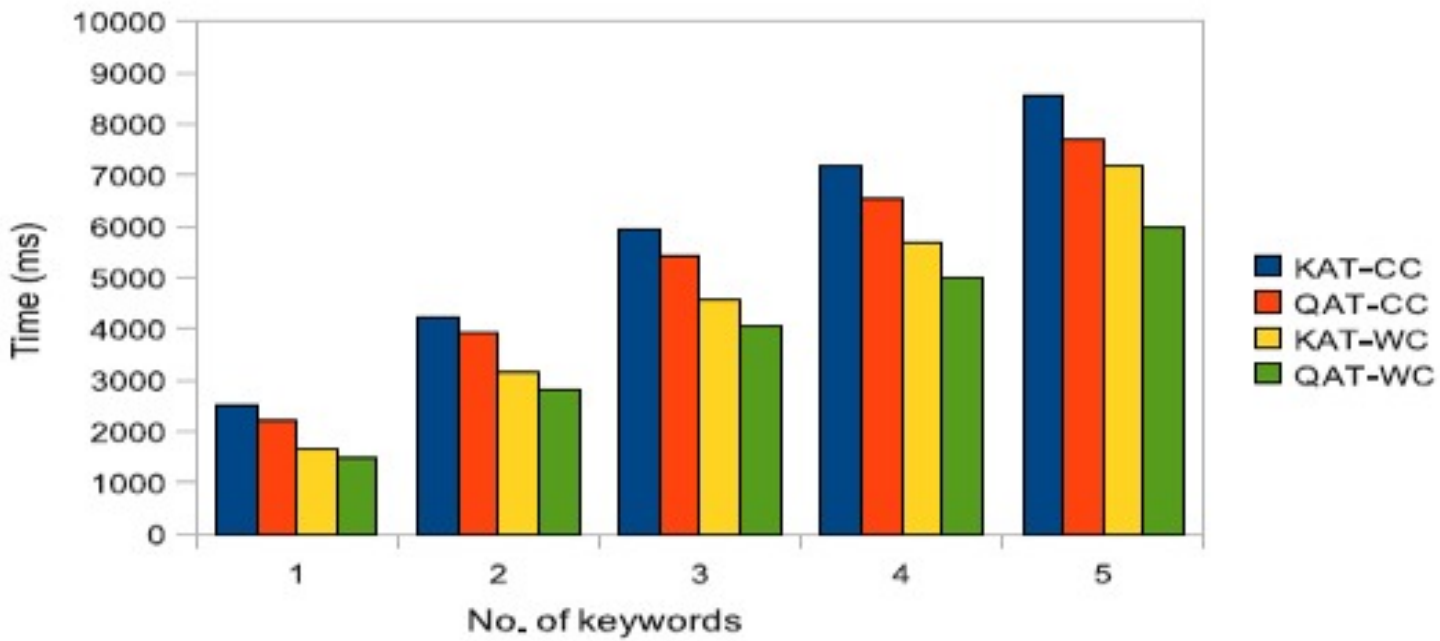



- Set of 5 keywords
  - for N < 5 keywords, avg of all subsets of size N
- Cold cache: restart DB, flush file system cache
- Recommend flash storage for best performance

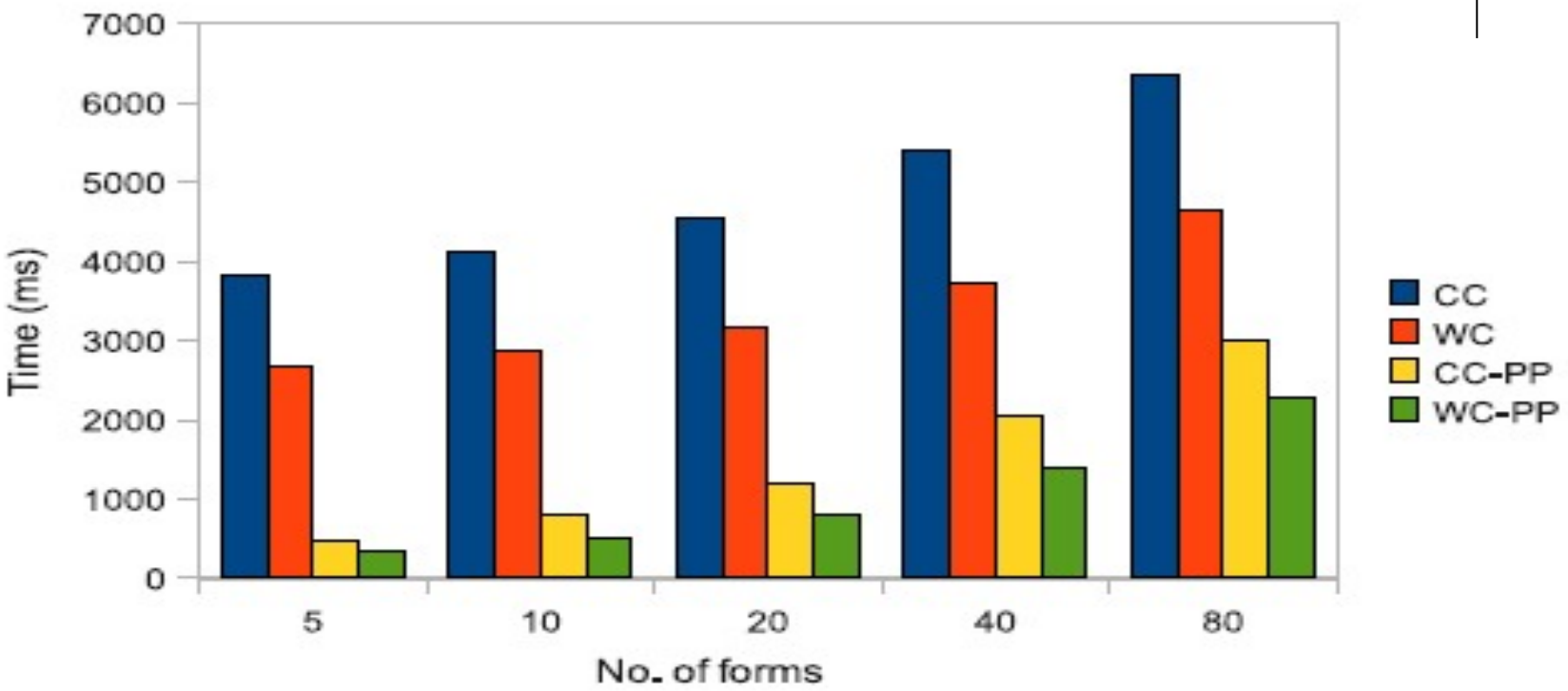
# Keyword Performance: KAT vs QAT



- KAT vs QAT: QAT slightly faster

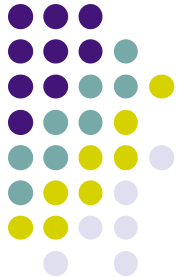


# Scalability With #Forms



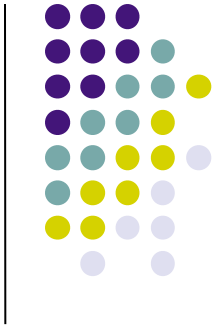

- Sublinear scaling with #forms
- Pruning optimization: eliminate query if some keyword is not present in any of its relations
  - Works very well

# Form Result Materialization



- Overheads of form materialization approach
  - Implemented incremental view maintenance for form queries on updates to underlying relations
  - Time overhead of 1 second on flash for adding course registrations, which normally takes 10s of msecs.
    - Unacceptable at peak load
  - Space overhead: 1.4 GB extra for 1 GB academic database
  - Hard to incrementally maintain some queries
- Our approach has no overheads on normal operation

# Conclusion



- Our techniques support efficient keyword search on Web applications
  - Without any intrusive changes to application
  - Practical, and works especially well with flash disk
- Future work
  - Better ranking functions, customized to user
  - Global fulltext index on all tables to reduce seeks
  - Larger class of queries (e.g. top-K, case statements)
  - Conditional query execution (branches in application)
  - Automated analysis of applications to extract form queries
  - Integration with access control
    - Implemented in our prototype, but need to generalize

# Screenshot of Query Result

| #. | Id. | FormUrl | Parameters (#params) | Score |
|---|---|---|---|---|
| 1> | 6. | Provisional Certficate<br>Provisional Certificate<br>*Time: 35 ms.* | [rollno] (1) | 1.0 |
| 2> | 22. | Print Students permanent address<br>Print students permanent address department-program wise along with an option of printing for a particular convocation<br>*Time: 25 ms.* | [rollno] (1) | 1.0 |
| 3> | 46. | Senate Sheet<br>Report showing the information sent to Senate<br>*Time: 36 ms.* | [txtRollno] (1) | 4.0 |
| 4> | 33. | Students for whom grades have NOT been received for a particular course<br>View/Print list of all students for whom grades have NOT been received for a particular course in the specified acadyear and sem<br>*Time: 44 ms.* | [ayr, sem, dept, prog] (1) | 7.0 |
| 5> | 23. | Print Students permanent address | [batchyr, dept, prog] (1) | 15.0 |