Keyword Searches on Relational Databases

CPS 196.3
Introduction to Database Systems

Drawbacks of the BANKS approach

- Must compute and maintain a link graph of all database tuples
  - May be huge
- Assumes that the data graph fits in main memory
- Does not take advantage of the processing capabilities of the DBMS

DISCOVER

V. Hristidis and Y. Papakonstantinou. “DISCOVER:Keyword Search in Relational Databases.” In VLDB 2002
- Some following slides are borrowed from Hristidis’s VLDB talk

- No need to pre-compute the joins between tuples
- Construct a schema graph, not a data graph
- At search time, tuples containing keywords are joined together according to the schema graph to produce answers to a query

Result of a keyword query

Pretty much the same as BANKS

- Result is a tree of tuples, where
  - Each edge in the tree corresponds to a primary-foreign key relationship
  - Total: every keyword is the query is contained in some tuple in the tree
  - Minimal: no tuple in the tree is redundant
    - That is, removing it would result in a tree that does not satisfy the total property

Example database

Orders, Customer, Nation
- Order(orderkey) references Customer(customerkey)
- Customer(customerkey) references Nation(nationkey)

Here edges go from primary key to foreign key
Example query

Query: Smith + Miller

<table>
<thead>
<tr>
<th>ORDERS</th>
<th>CUSTOMER</th>
<th>NATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERKEY</td>
<td>CUSTKEY</td>
<td>TOTALPRICE</td>
</tr>
<tr>
<td>10001</td>
<td>12312</td>
<td>$5,000</td>
</tr>
<tr>
<td>10001</td>
<td>12312</td>
<td>$3,000</td>
</tr>
<tr>
<td>10001</td>
<td>10002</td>
<td>$8,000</td>
</tr>
</tbody>
</table>

Results:  
- \( O \leftarrow C \rightarrow O Miller \) (size: 2)  
- \( O \leftarrow C \leftrightarrow N \rightarrow C \rightarrow O Miller \) (size: 4)  
- Smaller sizes usually mean tighter association between keywords

Candidate network examples

Given \( O^{Smith} \) and \( O^{Miller} \), what are the possible joins?  
- CN1: \( O^{Smith} \leftarrow C \rightarrow O^{Miller} \) (size: 2)  
- CN2: \( O^{Smith} \leftarrow C \leftarrow N \rightarrow C \rightarrow O^{Miller} \) (size: 4)  
- Not minimal; no need to consider  
- CN4: \( O^{Smith} \leftarrow C \rightarrow O \leftarrow C \rightarrow O^{Miller} \)  
  - O is redundant because the two Cs must be the same  
  - Pruning condition: no need to consider \( R' \rightarrow S \leftarrow R' \)

Number of candidate networks

Might be data-bound if  
- A table contains two foreign key references  
- There is a cycle of primary-foreign key relationships

Candidate networks generator

- Traverse tuple set graph breadth-first  
- \( Q \leftarrow \) tuple sets containing keyword \( k_i \)  
- For each network \( x \) in \( Q \) do  
  - If \( x \) should be pruned, remove \( x \) from \( Q \)  
  - Else if \( x \) is a valid candidate network output \( x \)  
  - Else expand \( x \) by one tuple set in all possible directions in the tuple set graph, and insert the expansions to \( Q \)  
  - Example: if \( x = O^{Smith} \leftarrow C \), then we add to \( Q \):  
    - \( O^{Smith} \leftarrow C \rightarrow O^{Miller} \)  
    - \( O^{Smith} \leftarrow C \leftarrow N \rightarrow C \rightarrow O^{Miller} \)  
- This algorithm produces a complete and non-redundant set of candidate networks

Execution plan

Each candidate network corresponds to a join  
- Candidate networks:  
  - CN1: \( O^{Smith} \leftarrow C \rightarrow O^{Miller} \)  
  - CN2: \( O^{Smith} \leftarrow C \leftarrow N \rightarrow C \rightarrow O^{Miller} \)  
- Execution plan:  
  - CN1: \( O^{Smith} \leftarrow C \rightarrow O^{Miller} \)  
  - CN2: \( O^{Smith} \leftarrow C \leftarrow N \rightarrow C \rightarrow O^{Miller} \)  
- An optimized plan that reuses common subexpressions:  
  - Temp \( \leftarrow O^{Miller} \leftarrow C \)  
  - CN1: \( \text{Temp} \leftarrow O^{Miller} \)  
  - CN2: \( \text{Temp} \leftarrow N \leftarrow C \rightarrow O^{Miller} \)
Optimizing the execution plan

- Optimal reuse of common subexpressions is NP-complete
- Greedy algorithm: pick a join (to compute as intermediate result) that maximizes frequency / log(size)
  - $0 \leq a \leq 1$ favors reusability
  - $0 \leq b \leq 1$ favors small intermediate results
  - Experiments show that $a = 1$ and $0 \leq b \leq 0.3$ yield good results

Conclusion

- Formalized what constitutes a complete answer to a keyword search query
- Provided an algorithm that enumerates all joins that could contribute to the answer
- Provided a heuristic for optimizing these joins to take advantage of common subexpressions

- But users usually do not need complete answers; they just want the “top” $k$ answers!
- Can tuples be related through relationships other than primary-foreign key relationship?