RankSQL: Query Algebra and Optimization for Relational Top-k Queries

Chengkai Li (UIUC)

joint work with

Kevin Chen-Chuan Chang (UIUC)
Ihab F. Ilyas (U. of Waterloo)
Sumin Song (UIUC)
Ranking (Top-k) Queries

Ranking is an important functionality in many real-world database applications:

- **E-Commerce, Web Sources**
  Find the *best* hotel deals by price, distance, etc.
- **Multimedia Databases**
  Find the *most similar* images by color, shape, texture, etc.
- **Text Retrieval, Search Engine**
  Find the *most relevant* records/documents/pages.
- **OLAP, Decision Support**
  Find the *top profitable* customers to send ads.
Example: Trip Planning

Suggest a hotel to stay and a museum to visit:

Select * 
From 
  Hotel h, Museum m 
Where 
  h.star=3 AND h.area=m.area 
Order By 
  cheap(h.price) + close(h.addr, "BWI airport") + related(m.collection,"dinosaur") 
Limit 5

<table>
<thead>
<tr>
<th>hotel</th>
<th>museum</th>
<th>cheap</th>
<th>close</th>
<th>related</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>m2</td>
<td>0.9</td>
<td>0.7</td>
<td>0.8</td>
<td>2.4</td>
</tr>
<tr>
<td>h2</td>
<td>m1</td>
<td>0.6</td>
<td>0.8</td>
<td>0.9</td>
<td>2.3</td>
</tr>
<tr>
<td>h1</td>
<td>m3</td>
<td>0.9</td>
<td>0.7</td>
<td>0.6</td>
<td>2.2</td>
</tr>
</tbody>
</table>

membership dimension: 
Boolean predicates, 
Boolean function

order dimension: 
ranking predicates, 
monotonic scoring function
Processing Ranking Queries in Traditional RDBMS

Select *
From Hotel h, Museum m
Where
h.area = m.area
σ h.star = 3
Order By
cheap + close + related
Limit 5

5 results

scan(h)
scan(m)

h.area = m.area
σ h.star = 3
Problems of Traditional Approach

• Naïve *Materialize-then-Sort* scheme
• Overkill:
  total order of all results;
  only 5 top results are requested.
• Very inefficient:
  – Scan large base tables;
  – Join large intermediate results;
  – Evaluate every ranking on every tuple;
  – Full sorting.
Therefore the problem is:

Unlike Boolean constructs, ranking is second class.

– Ranking is processed as a Monolithic component ($R$), always after the Boolean component ($B$).
How did we make Boolean “first class”?

Select *
From
  Hotel h, Museum m
Where
  h.star=3 AND h.area=m.area

(1) *materialize-then-filter*
First Class: Splitting and Interleaving

Select *
From Hotel h, Museum m
Where h.star=3 AND h.area=m.area

(2) $B$ is split into joins and selections, which *interleave* with each other.

(1) *materialize-then-filter*

\[ \sigma_{h.area=m.area} \]

\[ h.area=m.area \]

\[ \sigma_{h.star=3} \]

scan(h)

scan(m)
Ranking Query Plan

**Materialize-then-sort:**
naïve, overkill

**Split and Interleave:**
reduction of intermediate results, thus processing cost
Possibly orders of magnitude improvement

Implementation in PostgreSQL

plan1: traditional materialize-then-sort plan
plan2-4: new ranking query plans

Observations:
an extended plan space with plans of various costs.
RankSQL

• Goals:
  – Support ranking as a first-class query type in RDBMS; splitting ranking.
  – Integrate ranking with traditional Boolean query constructs. interleaving ranking with other operations.

• Foundation: Rank-Relational Algebra
  – data model: rank-relation
  – operators: new and augmented
  – algebraic laws

• Query engine:
  – executor: physical operator implementation
  – optimizer: plan enumeration, cost estimation
Two Logical Properties of Rank-Relation

- **Membership** of the tuples: evaluated Boolean predicates
- **Order** among the tuples: evaluated ranking predicates

**Membership** (\(\mathcal{B}\))
- \(h.\text{area}=m.\text{area}, \ h.\text{star}=3\)

**Order** (\(\mathcal{R}\))
- \(\text{close}(h.\text{addr}, \ “\text{BWI airport”})\)
- \(\text{related}(m.\text{collection}, \ “\text{dinosaur”})\)
- \(\text{cheap}(h.\text{price})\)
Ranking Principle: what should be the order?

F = cheap + close + related

<table>
<thead>
<tr>
<th>hotel</th>
<th>cheap</th>
<th>upper bound</th>
<th>museum</th>
<th>close</th>
<th>related</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>0.9</td>
<td>2.9</td>
<td>*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h2</td>
<td>0.6</td>
<td>2.6</td>
<td>*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Upper-bound** determines the order:
- Without further processing h1, we cannot output any result;
Ranking Principle: upper-bound determines the order

F = cheap + close + related

<table>
<thead>
<tr>
<th>hotel</th>
<th>cheap</th>
<th>upper bound</th>
<th>museum</th>
<th>close</th>
<th>related</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>0.9</td>
<td>2.9</td>
<td>*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h2</td>
<td>0.6</td>
<td>2.6</td>
<td>*</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Upper-bound** determines the order:
- Without further processing h1, we cannot output any result;
- Processing in the “promising” order, avoiding unnecessary processing.
Rank-Relation

• *Rank-relation* $R_P^F$
  
  R: relation
  
  F: monotonic scoring function over predicates $(p_1, \ldots, p_n)$
  
  $P \subseteq \{p_1, \ldots, p_n\}$: evaluated predicates

• **Logical Properties:**
  
  – **Membership:**
    
    R (as usual)
  
  – **Order:** <
    
    $\forall t_1, t_2 \in R_P^F: t_1 < t_2$ iff $F_P[t_1] < F_P[t_2]$. (by upper-bound)
Operators

To achieve splitting and interleaving:

• **New operator:**
  – \( \mu \): *evaluate ranking predicates piece by piece.*
    implementation: MPro (Chang et al. SIGMOD02).

• **Extended operators:**
  – rank-selection
  – rank-join
    implementation: HRJN (Ilyas et al. VLDB03).
  – rank-scan
  – rank-union, rank-intersection.
Example

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Example

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Example

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Example

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Select * 
From Hotel H 
Order By p1+p2+p3 
Limit 1
Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Select *
From Hotel H
Order By p1+p2+p3
Limit 1
### Example

<table>
<thead>
<tr>
<th>hotel</th>
<th>...</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td></td>
<td>0.7</td>
<td>0.8</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>h2</td>
<td></td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.55</td>
</tr>
<tr>
<td>h3</td>
<td></td>
<td>0.7</td>
<td>1.0</td>
<td>1.0</td>
<td>2.7</td>
</tr>
<tr>
<td>h4</td>
<td></td>
<td>0.7</td>
<td>1.0</td>
<td>1.0</td>
<td>2.7</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0.7</td>
<td>1.0</td>
<td>1.0</td>
<td>2.7</td>
</tr>
</tbody>
</table>

Select *
From Hotel H
Order By p1+p2+p3
Limit 1

<table>
<thead>
<tr>
<th>hotel</th>
<th>upper-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>2.55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hotel</th>
<th>upper-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>2.75</td>
</tr>
<tr>
<td>h1</td>
<td>2.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hotel</th>
<th>upper-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>2.9</td>
</tr>
<tr>
<td>h1</td>
<td>2.7</td>
</tr>
</tbody>
</table>
### Example

```
<table>
<thead>
<tr>
<th>hotel</th>
<th>...</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td></td>
<td>0.7</td>
<td>0.8</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>h2</td>
<td></td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.55</td>
</tr>
<tr>
<td>h3</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>h4</td>
<td></td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
</tbody>
</table>
```

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Example

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Select *  
From Hotel H  
Order By p1+p2+p3  
Limit 1

<table>
<thead>
<tr>
<th>hotel</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>2.4</td>
</tr>
<tr>
<td>h2</td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.55</td>
</tr>
<tr>
<td>h3</td>
<td>0.5</td>
<td>0.45</td>
<td>1.0</td>
<td>1.95</td>
</tr>
<tr>
<td>h4</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
<tr>
<td>...</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td>2.5</td>
</tr>
</tbody>
</table>

**Example**

Scan \(_{p_1}(H)\)

\[ R_{p_1} \]

\[ R_{p_1+p_2} \]

\[ R_{p_1+p_2+p_3} \]

<table>
<thead>
<tr>
<th>hotel</th>
<th>upper-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>2.55</td>
</tr>
<tr>
<td>h1</td>
<td>2.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hotel</th>
<th>upper-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>2.75</td>
</tr>
<tr>
<td>h1</td>
<td>2.5</td>
</tr>
<tr>
<td>h3</td>
<td>1.95</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hotel</th>
<th>upper-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>2.9</td>
</tr>
<tr>
<td>h1</td>
<td>2.7</td>
</tr>
<tr>
<td>h3</td>
<td>2.5</td>
</tr>
</tbody>
</table>
In contrast: materialize-then-sort

<table>
<thead>
<tr>
<th>hotel</th>
<th>...</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td></td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>2.4</td>
</tr>
<tr>
<td>h2</td>
<td></td>
<td>0.9</td>
<td>0.85</td>
<td>0.8</td>
<td>2.55</td>
</tr>
<tr>
<td>h3</td>
<td></td>
<td>0.5</td>
<td>0.45</td>
<td>0.75</td>
<td>1.7</td>
</tr>
<tr>
<td>h4</td>
<td></td>
<td>0.4</td>
<td>0.7</td>
<td>0.95</td>
<td>2.05</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Select *
From Hotel H
Order By p1+p2+p3
Limit 1
Impact of Rank-Relational Algebra

- Ranking Query
- parser, rewriter
- Logical Query Plan
  - optimizer
  - cost model
  - Plan enumerator
- Physical Query Plan
- executor
  - operators and algebraic laws
  - implementation of operators
Optimization

• Two-dimensional enumeration:
  ranking (ranking predicate scheduling)
  and
  filtering (join order selection)

• Sampling-based cardinality estimation
Two-Dimensional Enumeration

- (1 table, 0 predicate)
  \(\text{seqScan}(H), \ \text{idxScan}(H), \ \text{seqScan}(M), \ldots\)

- (1 table, 1 predicate)
  \(\text{rankScan}_{\text{cheap}}(H), \ \mu_{\text{cheap}}(\text{seqScan}(H)), \ldots\)

- (1 table, 2 predicates)
  \(\mu_{\text{close}}(\text{rankScan}_{\text{cheap}}(H)), \ldots\)

- (2 table, 0 predicate)
  \(\text{NestLoop}(\text{seqScan}(H), \text{seqScan}(M)), \ldots\)

- (2 table, 1 predicate)
  \(\text{NRJN}(\text{rankScan}_{\text{cheap}}(H), \text{seqScan}(M)), \ldots\)

- and so on…
Related Work

• Middleware
  Fagin et al. (PODS 96,01), Nepal et al. (ICDE 99), Günter et al. (VLDB 00), Bruno et al. (ICDE 02), Chang et al. (SIGMOD 02)

• RDBMS, outside the core
  Chaudhuri et al. (VLDB 99), Chang et al. (SIGMOD 00), Hristidis et al. (SIGMOD 01), Tsaparas et al. (ICDE 03), Yi et al. (ICDE 03)

• RDBMS, in the query engine
  – Physical operators and physical properties
    Carey et al. (SIGMOD 97), Ilyas et al. (VLDB 02, 03, SIGMOD 04), Natsev et al. (VLDB 01)
  – Algebra framework
    Chaudhuri et al. (CIDR 05)
Conclusion: RankSQL System

• Goal:
  Support ranking as a first-class query type;
  Integrate ranking with Boolean query constructs.

• Our approach:
  – Algebra: rank-relation,
    new and augmented rank-aware operators,
    algebraic laws
  – Optimizer: two-dimensional enumeration,
    sampling-based cost estimation

• Implementation: in PostgreSQL

Welcome to our demo in VLDB05!