Supporting Ad-Hoc Ranking Aggregates

Chengkai Li (UIUC)

joint work with

Kevin Chang (UIUC)    Ihab Ilyas (Waterloo)
Ranking (Top-k) Queries

Find the top $k$ answers with respect to a ranking function, which often is the aggregation of multiple criteria.

Ranking is important in many database applications:

- E-Commerce
  Find the best hotel deals by price, distance, etc.
- Multimedia Databases
  Find the most similar images by color, shape, texture, etc.
- Search Engine
  Find the most relevant records/documents/pages.
- OLAP, Decision Support
  Find the top profitable customers to send ads.
RankSQL: a RDBMS with Efficient Support of Ranking Queries

- **Rank-Aware Query Operators** [SIGMOD02, VLDB03]

- **Algebraic Foundation and Optimization Framework** [SIGMOD04, SIGMOD05]

  \[SPJ \text{ queries} \ (\text{SELECT} \ \cdots \ \text{FROM} \ \cdots \ \text{WHERE} \ \cdots \ \text{ORDER BY} \ \cdots)\]

- **Ad-Hoc Ranking Aggregate Queries** [SIGMOD06]

  top \(k\) groups instead of tuples.
  \((\text{SELECT} \ \cdots \ \text{FROM} \ \cdots \ \text{WHERE} \ \cdots \ \text{GROUP BY} \ \cdots \ \text{ORDER BY} \ \cdots)\)
Example 1: Advertising an insurance product

What are the top 5 areas to advertise a new insurance product?

SELECT zipcode, 
   AVG(income*w1+age*w2+credit*w3) as score 
FROM customer 
WHERE occupation='student' 
GROUP BY zipcode 
ORDER BY score 
LIMIT 5
Example 2: Finding the most profitable combinations

- What are the 5 most profitable pairs of (product category, sales area)?

```sql
SELECT P.category, S.zipcode, 
       MID_SUM(S.price - P.manufact_price) 
       as score 
FROM products P, sales S 
WHERE P.p_key=S.p_key 
GROUP BY P.category, S.zipcode 
ORDER BY score 
LIMIT 5
```
Ad-Hoc Ranking

Ranking Condition : \( F = G(T) \)

\[ e.g. \text{AVG} (\text{income} \cdot w_1 + \text{age} \cdot w_2 + \text{credit} \cdot w_3) \]
\[ \text{MID\_SUM} (\text{S.price} - \text{P.manufact\_price}) \]

- **G**: group-aggregate function
  - Standard (e.g., sum, avg)
  - User-defined (e.g., mid_sum)

- **T**: tuple-aggregate function
  - arbitrary expression
  - e.g., \text{AVG} (\text{income}\cdot w_1 + \text{age}\cdot w_2 + \text{credit}\cdot w_3), \quad w_1, w_2, w_3 \text{ can be any values.}
Why “Ad-Hoc”?

DSS applications are exploratory and interactive:

- Decision makers try out various ranking criteria
- Results of a query as the basis for further queries
- It requires efficient techniques for fast response
Existing Techniques

- **Data Cube / Materialized Views:**
  - pre-computation
    - The views may not be built for the G:
      - e.g., \textit{mid\_sum} cannot be derived from \textit{sum}, \textit{avg}, etc.
    - The views may not be built for the T:
      - e.g., \textit{a+b} does \textit{not help in doing a*b}, and vice versa.

- **Materialize-Group-Sort:**
  - from the scratch
Materialize-Group-Sort Approach

Select zipcode, \( \text{AVG}(\text{income} \times w1 + \text{age} \times w2 + \text{credit} \times w3) \) as score
From Customer
Where
\( \text{occupation} = \text{'student'} \)
Group By zipcode
Order By score
Limit 5
Problems of Materialize-Group-Sort

- **Overkill:**
  Total order of all groups, although only top 5 are requested.

- **Inefficient:**
  Full materialization (scan, join, grouping, sorting).

(a) Traditional query plan.
Can We Do Better?

Without any further info, full materialization is all that we can do.

- Can we do better:
  - What info do we need?
  - How to use the info?

(a) Traditional query plan.
RankAgg vs. Materialize-Group-Sort

Goal: minimize the number of tuples processed.
(Partial vs. full materialization)

(a) Traditional query plan.
(b) New query plan.

all tuples (materialized)

all groups (materialized)

sort
group

Boolean operators

rank- & group-aware operators

tuple x₁

tuple x₂

(group g₁

(group g₂

(incremental)
Orders of Magnitude Performance Improvement

![Graph showing execution time vs. number of groups for Traditional Plan and New Plan.](image-url)
The Principles of RankAgg

- **Can we do better?** Upper-Bound Principle: best-possible goal
  There is a certain minimal number of tuples to retrieve before we can stop.

- **What info do we need?** Upper-Bound Principle: must-have info
  A non-trivial upper-bound is a must. (e.g., +infinity will not save anything.)
  Upper-bound of a group indicates the best a group can achieve, thus tells us if it is going to make top-k or not.

- **How to use the info?**
  - Group-Ranking Principle: Process the most promising group first.
  - Tuple-Ranking Principle: Retrieve tuples in a group in the order of T.

- **Together: Optimal Aggregate Processing**
  minimal number of tuples processed.
Running Example

Select g, SUM(v)
From R
Group By g
Order By SUM(v)
Limit 1
Must-Have Information

Assumptions for getting a non-trivial upper-bound:

- We focus on a (large) class of max-bounded function: $F[g]$ can be obtained by applying $G$ over the maximal $T$ of $g$’s members.

- We have the size of each group. (Will get back to this.)

- We can obtain the maximal value of $T$. (In the example, $v \leq 1$.)
Example: Group-Ranking Principle

Process the most promising group first.

group-aware scan

agg

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_1 & 1 & .7 \\
\hline
r_5 & 1 & .9 \\
\hline
r_7 & 1 & .6 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_2 & 2 & .3 \\
\hline
r_4 & 2 & .4 \\
\hline
r_8 & 2 & .25 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_3 & 3 & .9 \\
\hline
r_6 & 3 & .7 \\
\hline
\end{array}
\]
Example: Group-Ranking Principle

Process the most promising group first.

\[
\begin{array}{|c|c|c|}
\hline
\text{action} & F[g_1] & F[g_2] & F[g_3] \\
\hline
\text{initial} & 3.0 & 3.0 & 2.0 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_1 & 1 & .7 \\
r_5 & 1 & .9 \\
r_7 & 1 & .6 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_2 & 2 & .3 \\
r_4 & 2 & .4 \\
r_8 & 2 & .25 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_3 & 3 & .9 \\
r_6 & 3 & .7 \\
\hline
\end{array}
\]
Example: Group-Ranking Principle

Process the most promising group first.

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_1]$</th>
<th>$F[g_2]$</th>
<th>$F[g_3]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_1, 1, .7)$</td>
<td>2.7</td>
<td>3.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_5$</td>
<td>1</td>
<td>.9</td>
</tr>
<tr>
<td>$r_7$</td>
<td>1</td>
<td>.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_2$</td>
<td>2</td>
<td>.3</td>
</tr>
<tr>
<td>$r_4$</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>$r_8$</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_3$</td>
<td>3</td>
<td>.9</td>
</tr>
<tr>
<td>$r_6$</td>
<td>3</td>
<td>.7</td>
</tr>
</tbody>
</table>
Example: Group-Ranking Principle

Process the most promising group first.

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_1]$</th>
<th>$F[g_2]$</th>
<th>$F[g_3]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_1, 1, .7)$</td>
<td>2.7</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_2, 2, .3)$</td>
<td>2.7</td>
<td>2.3</td>
<td>2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_5$</td>
<td>1</td>
<td>.9</td>
</tr>
<tr>
<td>$r_7$</td>
<td>1</td>
<td>.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_4$</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>$r_8$</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_3$</td>
<td>3</td>
<td>.9</td>
</tr>
<tr>
<td>$r_6$</td>
<td>3</td>
<td>.7</td>
</tr>
</tbody>
</table>
Example: Group-Ranking Principle

Process the most promising group first.

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_1]$</th>
<th>$F[g_2]$</th>
<th>$F[g_3]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_1, 1, .7)$</td>
<td>2.7</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_2, 2, .3)$</td>
<td>2.7</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_5, 1, .9)$</td>
<td>2.6</td>
<td>2.3</td>
<td>2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_7$</td>
<td>1</td>
<td>.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_4$</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>$r_8$</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_3$</td>
<td>3</td>
<td>.9</td>
</tr>
<tr>
<td>$r_6$</td>
<td>3</td>
<td>.7</td>
</tr>
</tbody>
</table>
Example: Group-Ranking Principle

Process the most promising group first.

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_1]$</th>
<th>$F[g_2]$</th>
<th>$F[g_3]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_1, 1, .7)$</td>
<td>2.7</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_2, 2, .3)$</td>
<td>2.7</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_5, 1, .9)$</td>
<td>2.6</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_7, 1, .6)$</td>
<td>2.2</td>
<td>2.3</td>
<td>2.0</td>
</tr>
</tbody>
</table>

$$\mathcal{F}^g, \mathcal{R}^g, \mathcal{R}^v$$
Example: Group-Ranking Principle

Process the most promising group first.

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_1]$</th>
<th>$F[g_2]$</th>
<th>$F[g_3]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_1, 1, .7)$</td>
<td>2.7</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_2, 2, .3)$</td>
<td>2.7</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_5, 1, .9)$</td>
<td>2.6</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_7, 1, .6)$</td>
<td>2.2</td>
<td>2.3</td>
<td>2.0</td>
</tr>
<tr>
<td>$(r_4, 2, .4)$</td>
<td>2.2</td>
<td>1.7</td>
<td>2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_8$</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_3$</td>
<td>3</td>
<td>.9</td>
</tr>
<tr>
<td>$r_6$</td>
<td>3</td>
<td>.7</td>
</tr>
</tbody>
</table>
Example: Tuple-Ranking Principle

Retrieves tuples within a group in the order of tuple-aggregate function T.

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_2$</td>
<td>2</td>
<td>.3</td>
</tr>
<tr>
<td>$r_4$</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>$r_8$</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

not in the order of R.v

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_4$</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>$r_2$</td>
<td>2</td>
<td>.3</td>
</tr>
<tr>
<td>$r_8$</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

in the order of R.v
Example: Tuple-Ranking Principle

Retrieve tuples within a group in the order of tuple-aggregate function \( T \).

<table>
<thead>
<tr>
<th>action</th>
<th>( \overline{F}[g_2] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_2 )</td>
<td>2</td>
<td>.3</td>
</tr>
<tr>
<td>( r_4 )</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>( r_8 )</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

not in the order of \( R.v \)

<table>
<thead>
<tr>
<th>TID</th>
<th>R.g</th>
<th>R.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_4 )</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>2</td>
<td>.3</td>
</tr>
<tr>
<td>( r_8 )</td>
<td>2</td>
<td>.25</td>
</tr>
</tbody>
</table>

in the order of \( R.v \)
Example: Tuple-Ranking Principle

Retrieve tuples within a group in the order of tuple-aggregate function $T$.

\[
\begin{array}{|c|c|}
\hline
\text{action} & \overline{F[g_2]} \\
\hline
\text{initial} & 3.0 \\
(r_2, 2, .3) & 2.3 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_4 & 2 & .4 \\
r_8 & 2 & .25 \\
\hline
\end{array}
\]

not in the order of $R.v$

\[
\begin{array}{|c|c|c|}
\hline
\text{TID} & \text{R.g} & \text{R.v} \\
\hline
r_2 & 2 & .3 \\
r_8 & 2 & .25 \\
\hline
\end{array}
\]

in the order of $R.v$
Example: Tuple-Ranking Principle

Retrieve tuples within a group in the order of tuple-aggregate function T.

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_2]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
</tr>
<tr>
<td>(r_2, 2, .3)</td>
<td>2.3</td>
</tr>
<tr>
<td>(r_4, 2, .4)</td>
<td>1.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>action</th>
<th>$F[g_2]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>3.0</td>
</tr>
<tr>
<td>(r_4, 2, .4)</td>
<td>1.2</td>
</tr>
<tr>
<td>(r_2, 2, .3)</td>
<td>1.0</td>
</tr>
</tbody>
</table>

TID | R.g | R.v
---|-----|-----
$r_8$ | 2   | .25

not in the order of R.v

in the order of R.v
Implementing the Principles: Obtaining Group Size

- **Sizes ready:**
  Though $G(T)$ is ad-hoc, the Boolean conditions are shared in sessions of decision making.

- **Sizes from materialized information:**
  Similar queries computed.

- **Sizes from scratch:**
  Pay as much as materialize-group-sort for the 1st query; amortized by the future similar queries.
Implementing the Principles: Group-Aware Plans

- Current iterator
  
  \[ \text{GetNext}(g) \]

- New iterator
  
  \[ \text{GetNext}(g) \]
Conclusions

- **Ranking Aggregate Queries**
  - Top-k groups
  - Ad-Hoc ranking conditions

- **RankAgg**
  - Principles
    - Upper-Bound, Group-Ranking, and Tuple-Ranking
  - Optimal Aggregate Processing
    - Minimal number of tuples processed
  - Significant performance gains, compared with materialize-group-sort.