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 queries (SELECT … FROM … WHERE … ORDER BY …)

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

  top k groups instead of tuples.
  (SELECT … FROM … WHERE … *GROUP BY …* ORDER BY …)

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)?

```
SELECT     P.cateogy, 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. AVG (income*w1+age*w2+credit*w3)*

MID_SUM (S.price - 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., AVG (income*w1+age*w2+credit*w3), w1, w2, w3 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., *mid_sum cannot be derived from sum, avg, etc.*
  - The views may not be built for the T:
    e.g., a+b *does not help in doing a*b, and vice versa.*

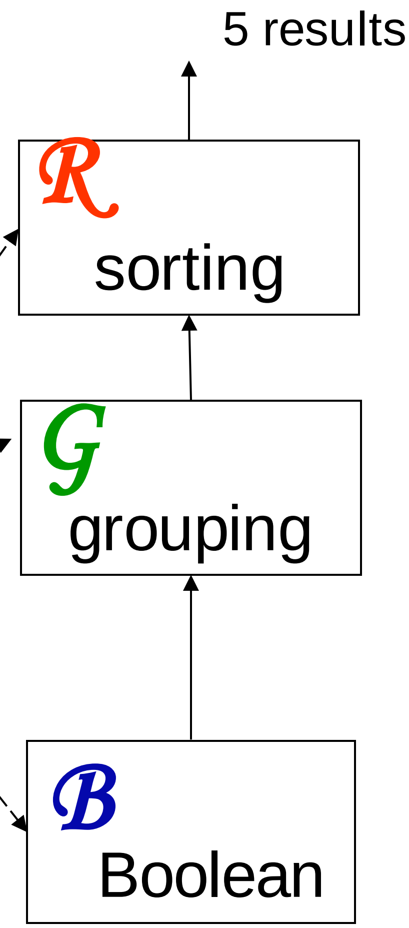
- **Materialize-Group-Sort:**
  from the scratch

Materialize-Group-Sort Approach

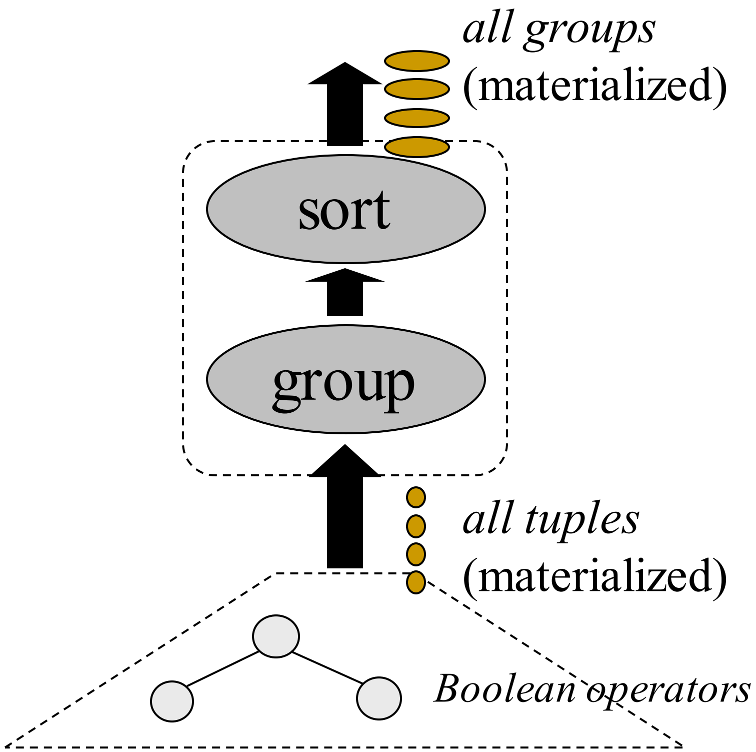
Select zipcode, *AVG(income*
**w1+age*w2+credit*w3)*
as score
From Customer
Where
occupation='student'

Group By zipcode

Order By **score**
Limit 5



Problems of Materialize-Group-Sort



(a) Traditional query plan.

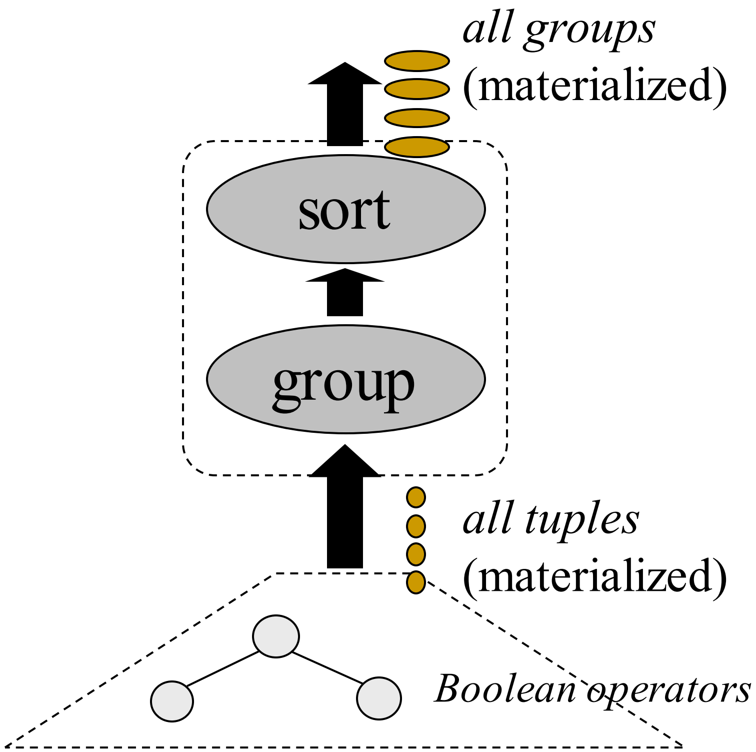
- Overkill:

  Total order of all groups, although only top 5 are requested.

- Inefficient:

  Full materialization (scan, join, grouping, sorting).

Can We Do Better?



(a) Traditional query plan.

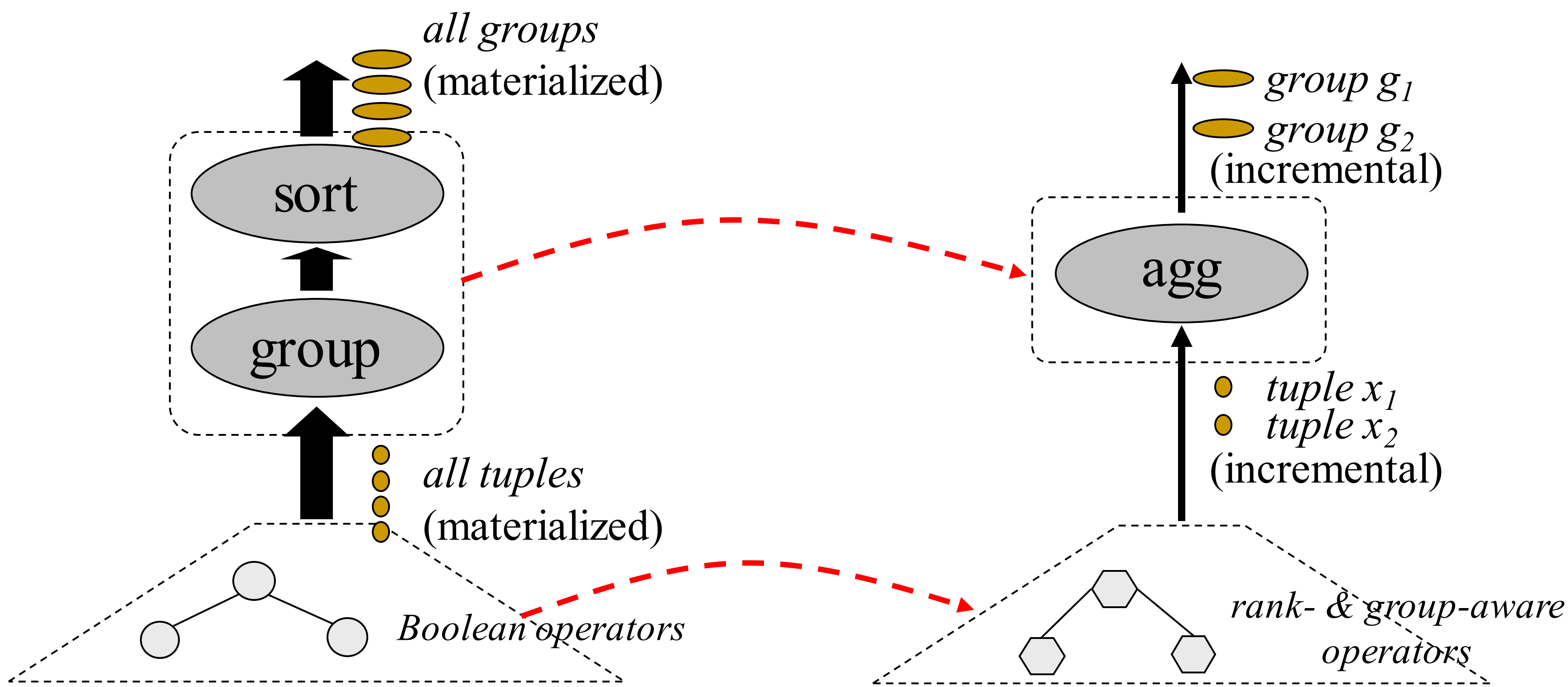
Without any further info, full materialization is all that we can do.

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

RankAgg vs. Materialize-Group-Sort

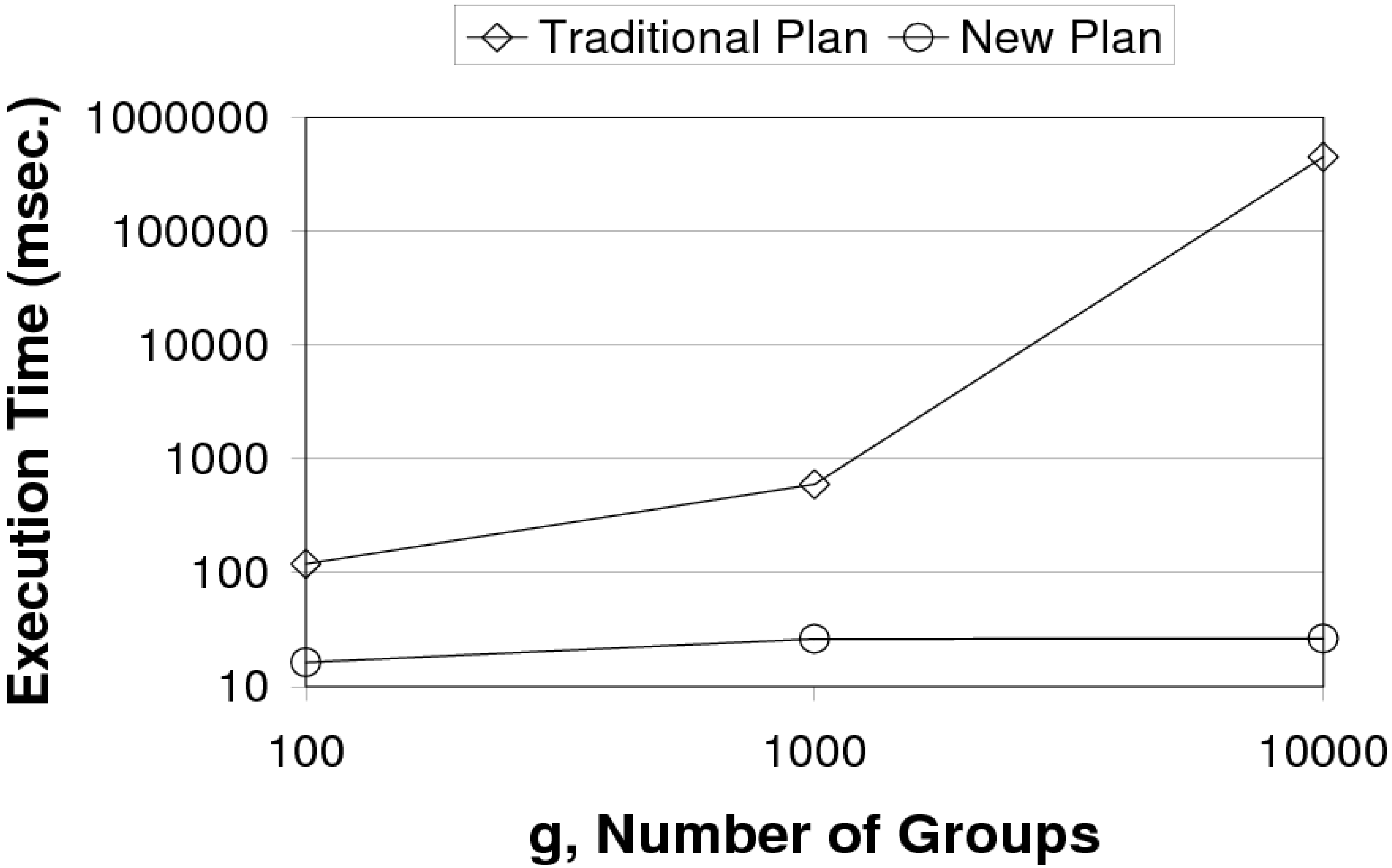
Goal: minimize the number of tuples processed.

(Partial vs. full materialization)



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

Orders of Magnitude Performance Improvement



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

TID	R.g	R.v
r_1	1	.7
r_2	2	.3
r_3	3	.9
r_4	2	.4
r_5	1	.9
r_6	3	.7
r_7	1	.6
r_8	2	.25

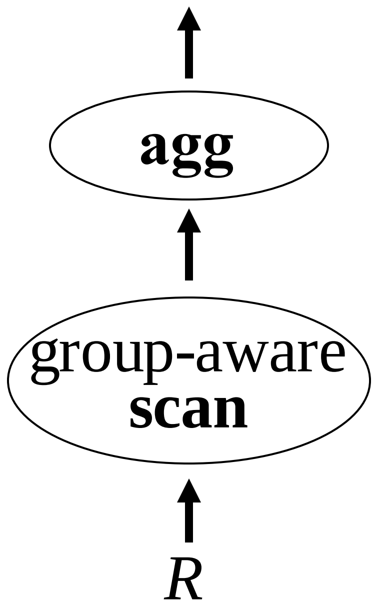
Must-Have Information

Assumptions for getting a non-trivial upper-bound:

- We focus on a (large) class of max-bounded function: *$\overline{F}$[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 <= 1.)

Example: Group-Ranking Principle

Process the most promising group first.



TID	R.g	R.v
r_1	1	.7
r_5	1	.9
r_7	1	.6

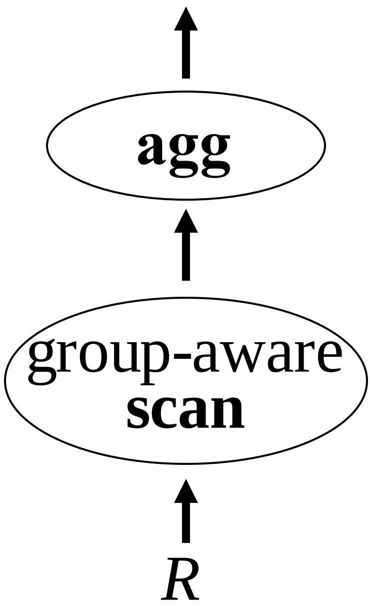
TID	R.g	R.v
r_2	2	.3
r_4	2	.4
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Group-Ranking Principle

Process the most promising group first.

action	$\overline{F}[g_1]$	$\overline{F}[g_2]$	$\overline{F}[g_3]$
initial	3.0	3.0	2.0



TID	R.g	R.v
r_1	1	.7
r_5	1	.9
r_7	1	.6

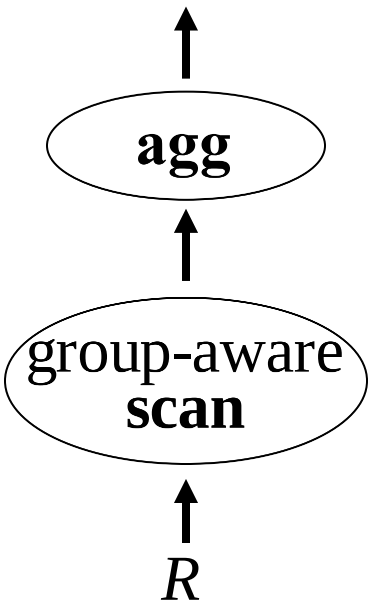
TID	R.g	R.v
r_2	2	.3
r_4	2	.4
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Group-Ranking Principle

Process the most promising group first.

action	$\overline{F}[g_1]$	$\overline{F}[g_2]$	$\overline{F}[g_3]$
initial	3.0	3.0	2.0
$(r_1, 1, .7)$	2.7	3.0	2.0



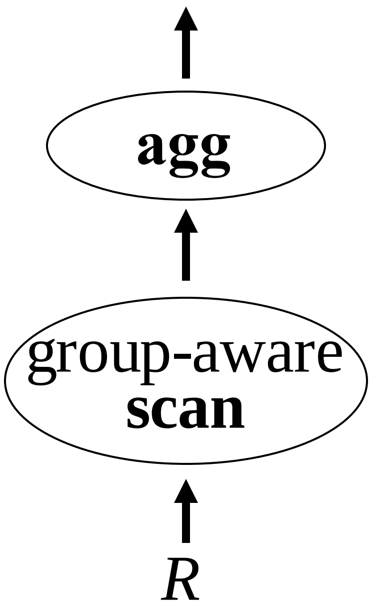
TID	R.g	R.v
r_5	1	.9
r_7	1	.6

TID	R.g	R.v
r_2	2	.3
r_4	2	.4
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Group-Ranking Principle

Process the most promising group first.



action	$\overline{F}[g_1]$	$\overline{F}[g_2]$	$\overline{F}[g_3]$
initial	3.0	3.0	2.0
(r_1, 1, .7)	2.7	3.0	2.0
(r_2, 2, .3)	2.7	2.3	2.0

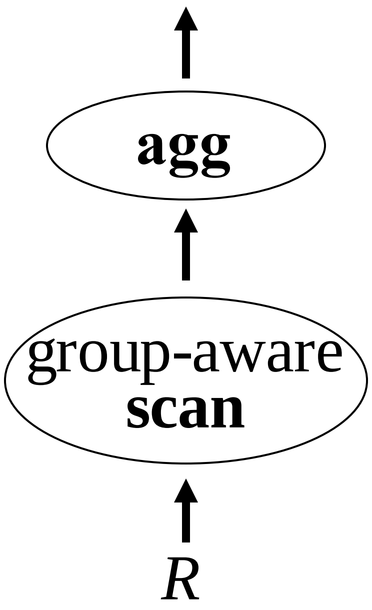
TID	R.g	R.v
r_5	1	.9
r_7	1	.6

TID	R.g	R.v
r_4	2	.4
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Group-Ranking Principle

Process the most promising group first.



action	$\overline{F}[g_1]$	$\overline{F}[g_2]$	$\overline{F}[g_3]$
initial	3.0	3.0	2.0
$(r_1, 1, .7)$	2.7	3.0	2.0
$(r_2, 2, .3)$	2.7	2.3	2.0
$(r_5, 1, .9)$	2.6	2.3	2.0

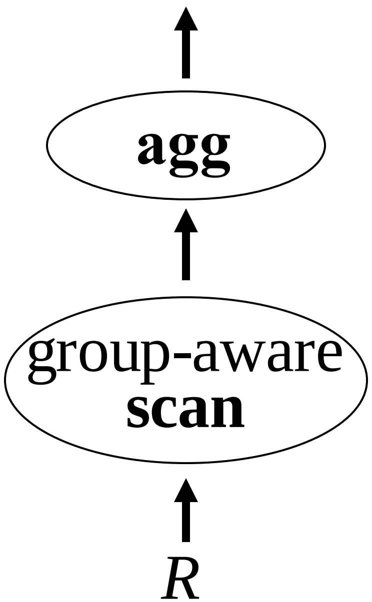
TID	R.g	R.v
r_7	1	.6

TID	R.g	R.v
r_4	2	.4
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Group-Ranking Principle

Process the most promising group first.



action	$\overline{F}[g_1]$	$\overline{F}[g_2]$	$\overline{F}[g_3]$
initial	3.0	3.0	2.0
$(r_1, 1, .7)$	2.7	3.0	2.0
$(r_2, 2, .3)$	2.7	2.3	2.0
$(r_5, 1, .9)$	2.6	2.3	2.0
$(r_7, 1, .6)$	2.2	2.3	2.0

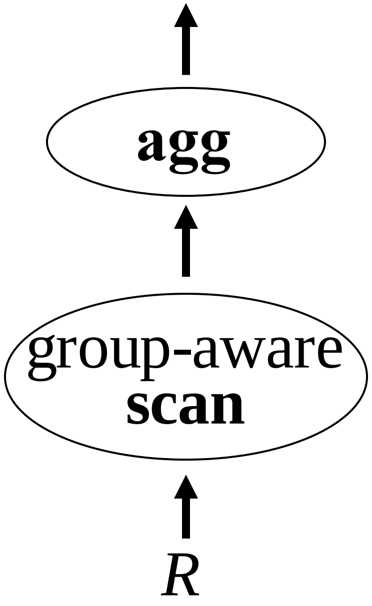
TID	R.g	R.v

TID	R.g	R.v
r_4	2	.4
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Group-Ranking Principle

Process the most promising group first.



action	$\overline{F}[g_1]$	$\overline{F}[g_2]$	$\overline{F}[g_3]$
initial	3.0	3.0	2.0
$(r_1, 1, .7)$	2.7	3.0	2.0
$(r_2, 2, .3)$	2.7	2.3	2.0
$(r_5, 1, .9)$	2.6	2.3	2.0
$(r_7, 1, .6)$	2.2	2.3	2.0
$(r_4, 2, .4)$	<u>2.2</u>	1.7	2.0

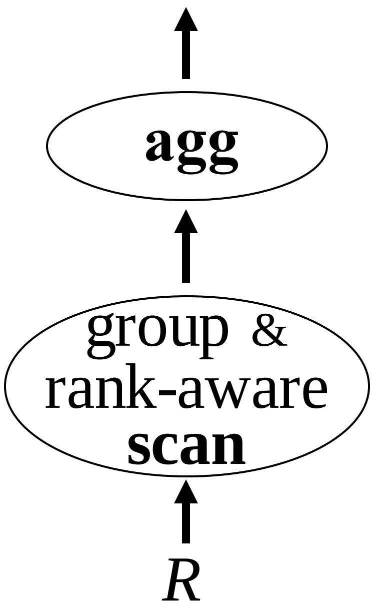
TID	R.g	R.v

TID	R.g	R.v
r_8	2	.25

TID	R.g	R.v
r_3	3	.9
r_6	3	.7

Example: Tuple-Ranking Principle

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



TID	R.g	R.v
r_2	2	.3
r_4	2	.4
r_8	2	.25

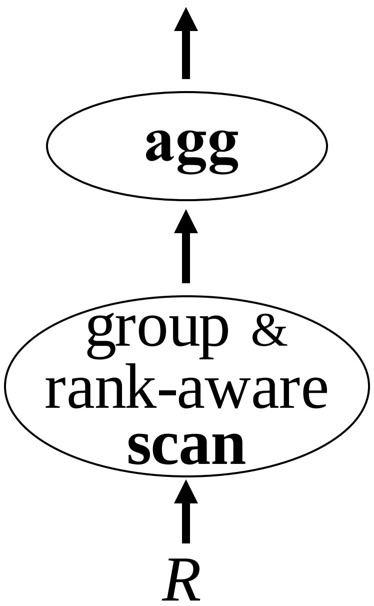
not in the order of R.v

TID	R.g	R.v
r_4	2	.4
r_2	2	.3
r_8	2	.25

in the order of R.v

Example: Tuple-Ranking Principle

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



action	$\overline{F}[g_2]$
initial	3.0

TID	R.g	R.v
r_2	2	.3
r_4	2	.4
r_8	2	.25

not in the order of R.v

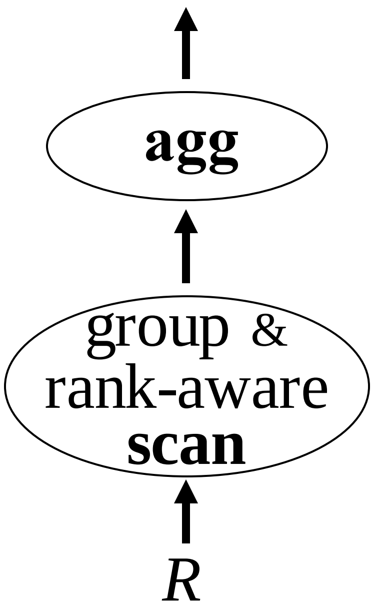
action	$\overline{F}[g_2]$
initial	3.0

TID	R.g	R.v
r_4	2	.4
r_2	2	.3
r_8	2	.25

in the order of R.v

Example: Tuple-Ranking Principle

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



action	$\overline{F}[g_2]$
initial	3.0
(r_2, 2, .3)	2.3

TID	R.g	R.v
r_4	2	.4
r_8	2	.25

not in the order of R.v

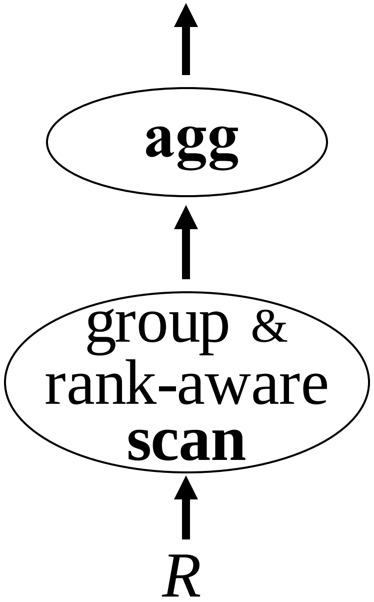
action	$\overline{F}[g_2]$
initial	3.0
(r_4, 2, .4)	1.2

TID	R.g	R.v
r_2	2	.3
r_8	2	.25

in the order of R.v

Example: Tuple-Ranking Principle

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



action	$\overline{F}[g_2]$
initial	3.0
(r_2, 2, .3)	2.3
(r_4, 2, .4)	1.7

TID	R.g	R.v
r_8	2	.25

not in the order of R.v

action	$\overline{F}[g_2]$
initial	3.0
(r_4, 2, .4)	1.2
(r_2, 2, .3)	1.0

TID	R.g	R.v
r_8	2	.25

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:

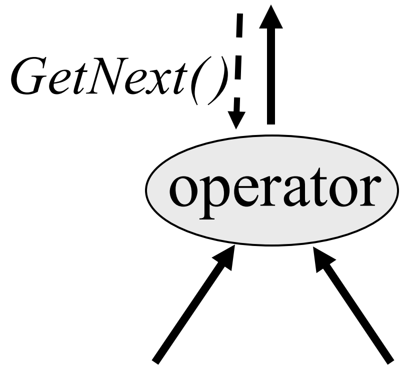
  Similar queries computed.

- Sizes from scratch:

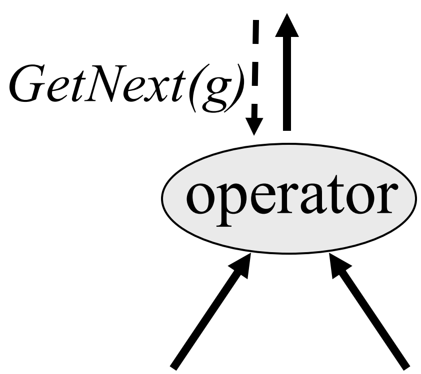
  Pay as much as materialize-group-sort for the 1st query; amortized by the future similar queries.

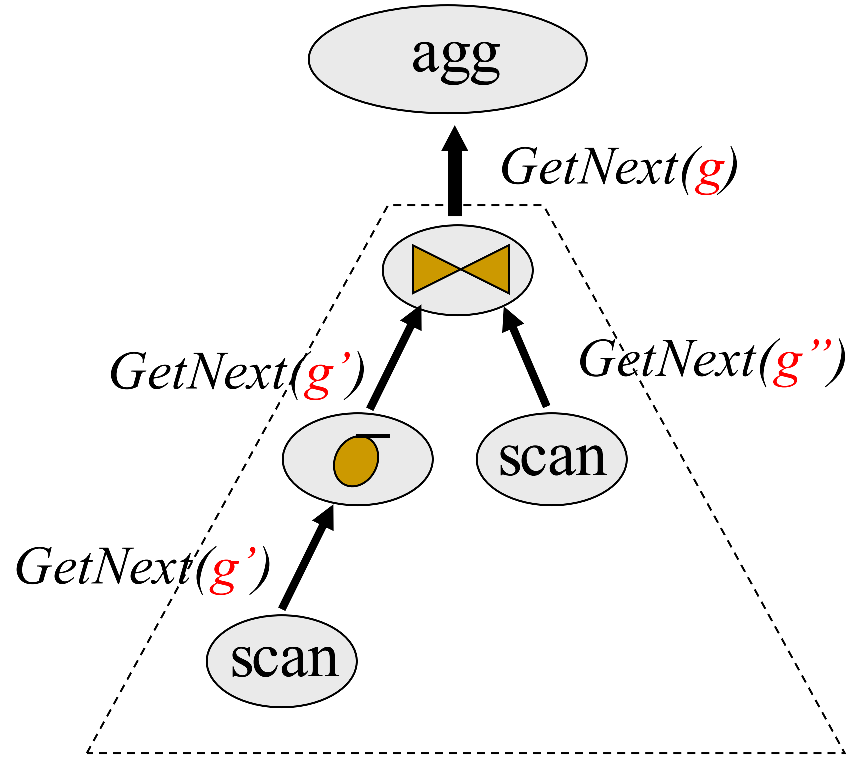
Implementing the Principles: Group-Aware Plans

- Current iterator



- New iterator





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.