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:

hotel	museum	cheap	close	related	score
h1	m2	0.9	0.7	0.8	2.4
h2	m1	0.6	0.8	0.9	2.3
h1	m3	0.9	0.7	0.6	2.2

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

$\mathcal{B}$ *membership dimension: Boolean predicates, Boolean function*

$\mathcal{R}$ *order dimension: ranking predicates, monotonic scoring function*

Processing Ranking Queries in Traditional RDBMS

Select *

From

  Hotel h, Museum m

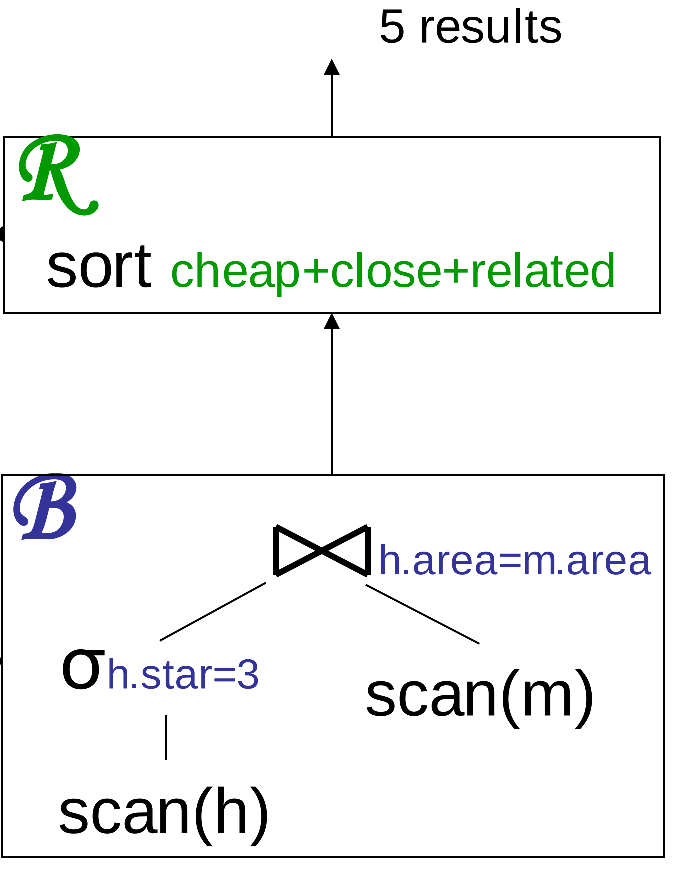
Where

  $\mathcal{B}$

Order By

  $\mathcal{R}$

Limit 5

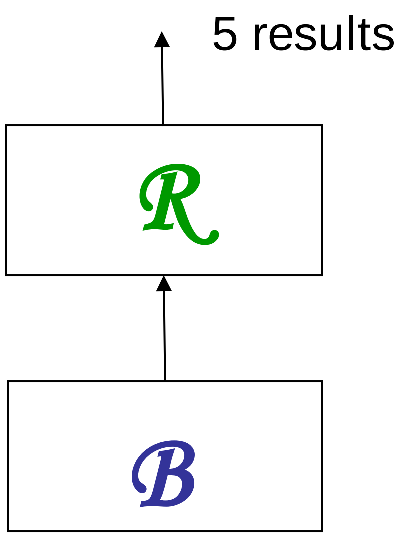


Problems of Traditional Approach

- 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.



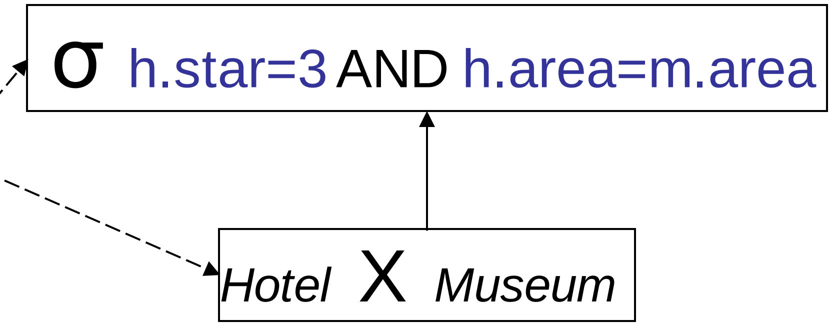
Therefore the problem is:

Unlike Boolean constructs, ranking is second class.

– Ranking is processed as a Monolithic component ($\mathcal{R}$), always after the Boolean component ($\mathcal{B}$).

How did we make Boolean "first class"?

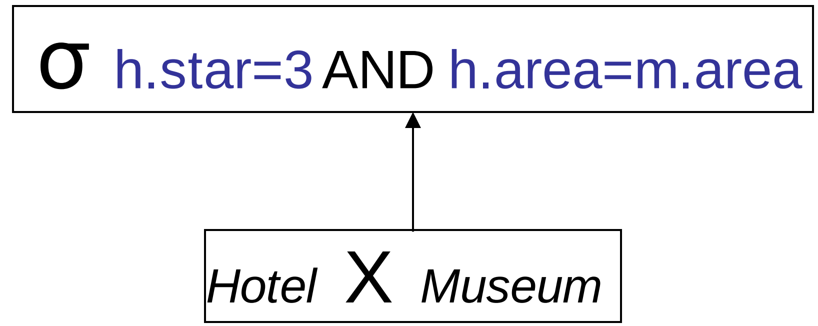
Select *
From
  Hotel h, Museum m
Where
  h.star=3 AND
  h.area=m.area



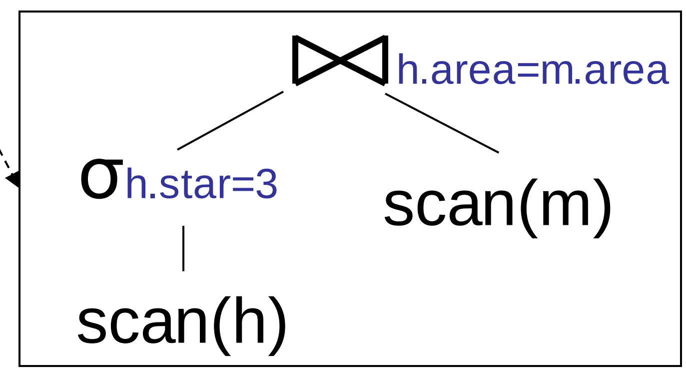
(1) *materialize-then-filter*

First Class: Splitting and Interleaving

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

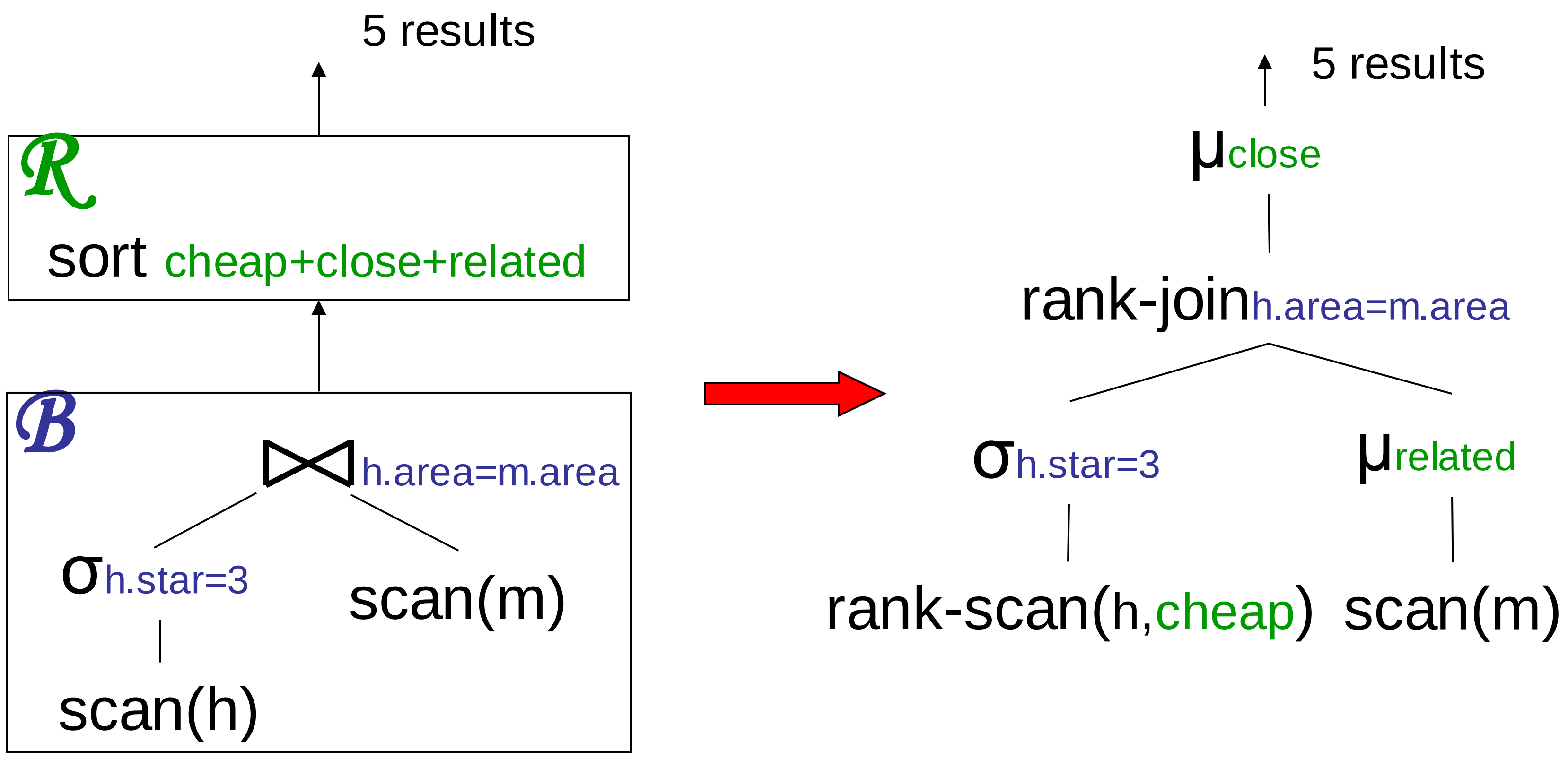


(1) *materialize-then-filter*



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

Ranking Query Plan



materialize-then-sort:
naïve, overkill

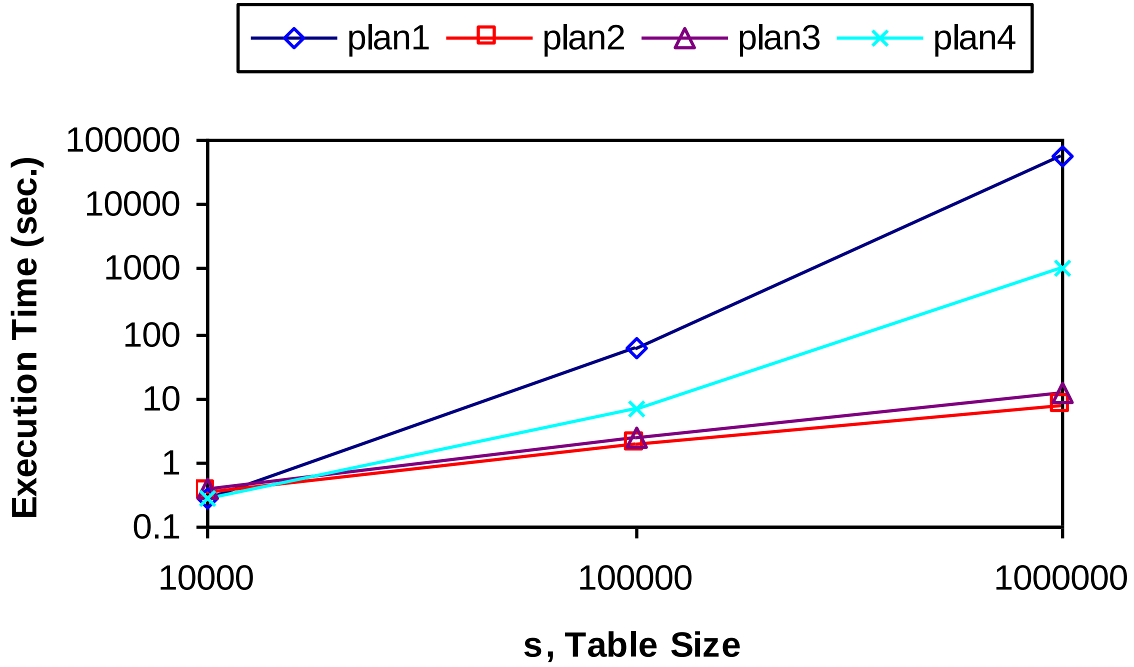
split and interleave:
reduction of intermediate results, thus processing cost

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

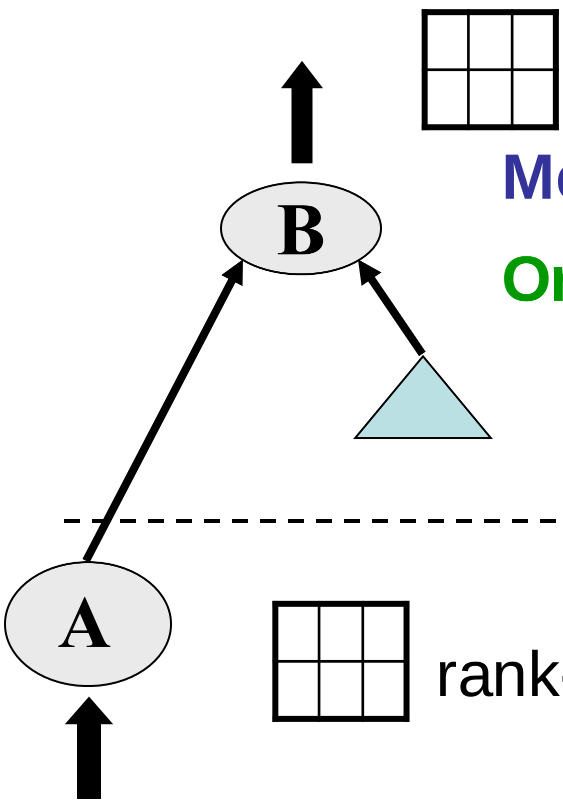
- Goals:
  - Support ranking as a first-class query type in RDBMS; splitting ranking.
  - Integrate ranking with traditional Boolean query constructs. interleaving ranking with other operations.

- 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 predciates



rank-relation

Membership($\mathcal{B}$): h.area=m.area, h.star=3

Order($\mathcal{R}$): close(h.addr, "BWI airport")
related(m.collection,"dinosaur")
cheap(h.price)

---

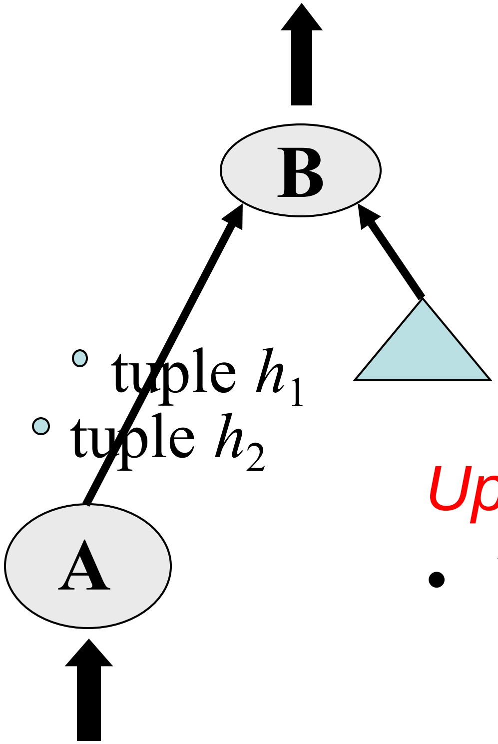
rank-relation

Membership($\mathcal{B}$): h.star=3

Order($\mathcal{R}$): cheap(h.price)

Ranking Principle: what should be the order?

F=cheap + close + related



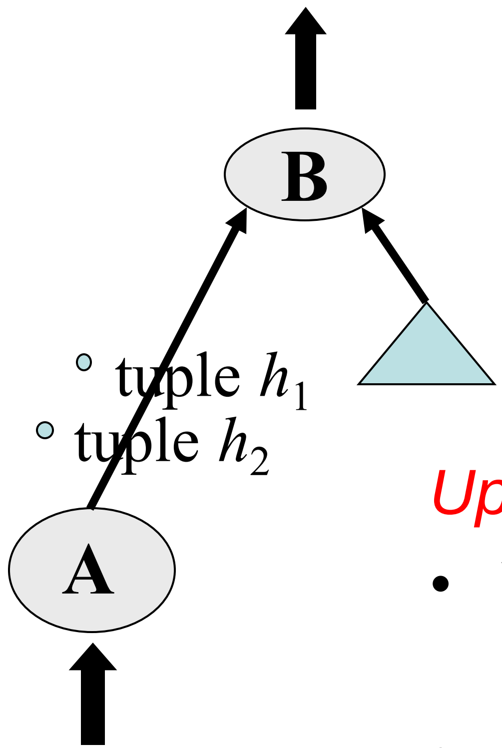
hotel	cheap	upper bound	museum	close	related
h1	0.9	2.9	*	1	1
h2	0.6	2.6	*	1	1
…	…	…	…	…	…

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



hotel	cheap	upper bound	museum	close	related
h1	0.9	2.9	*	1	1
h2	0.6	2.6	*	1	1
…	…	…	…	…	…

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 $(p_1, ..., p_n)$
  $P \subseteq \{p_1, . . . , p_n\}$: evaluated predicates

- Logical Properties:
  - Membership:
    R (as usual)
  - Order: <
    $\forall$ t1, t2 $\in R_P^F$: t1 < t2 iff $\overline{F}_P$ [t1] < $\overline{F}_P$ [t2].
    (by upper-bound)

Operators

To achieve splitting and interleaving:

- New operator:
  - ***μ: evaluate ranking predicates piece by piece.***

    implementation: MPro (Chang et al. SIGMOD02).

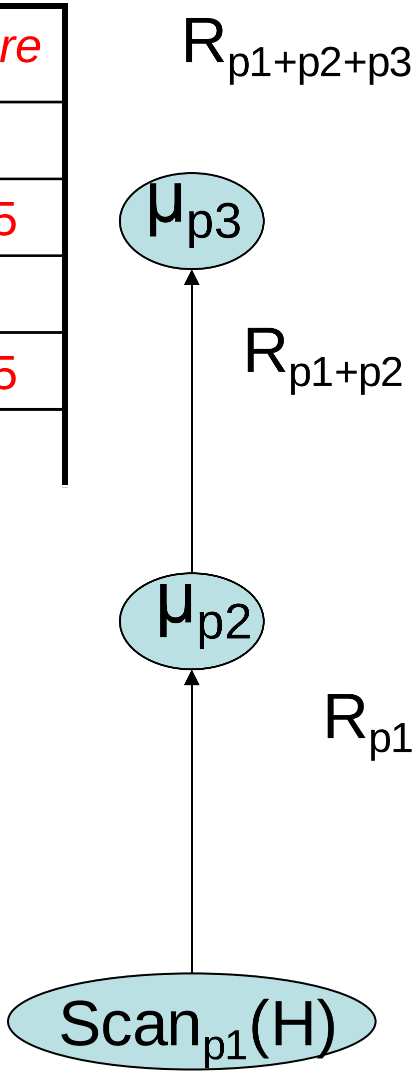
- Extended operators:
  - rank-selection
  - rank-join

    implementation: HRJN (Ilyas et al. VLDB03).
  - rank-scan
  - rank-union, rank-intersection.

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	0.9	2.4
h2		0.9	0.85	0.8	2.55
h3		0.5	0.45	0.75	1.7
h4		0.4	0.7	0.95	2.05
…		…	…	…	…

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



hotel	upper-bound

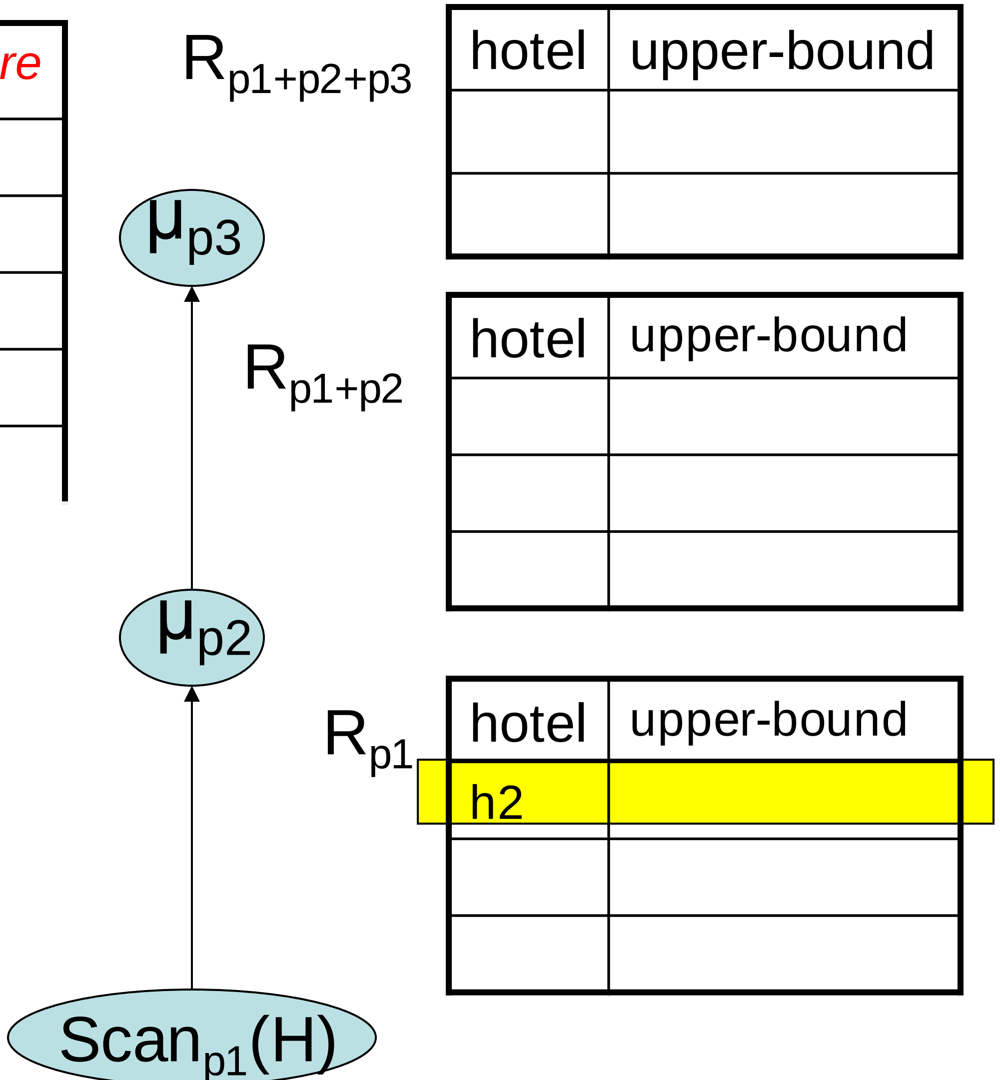
hotel	upper-bound

hotel	upper-bound

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1					
h2		0.9			
h3					
h4					
…					

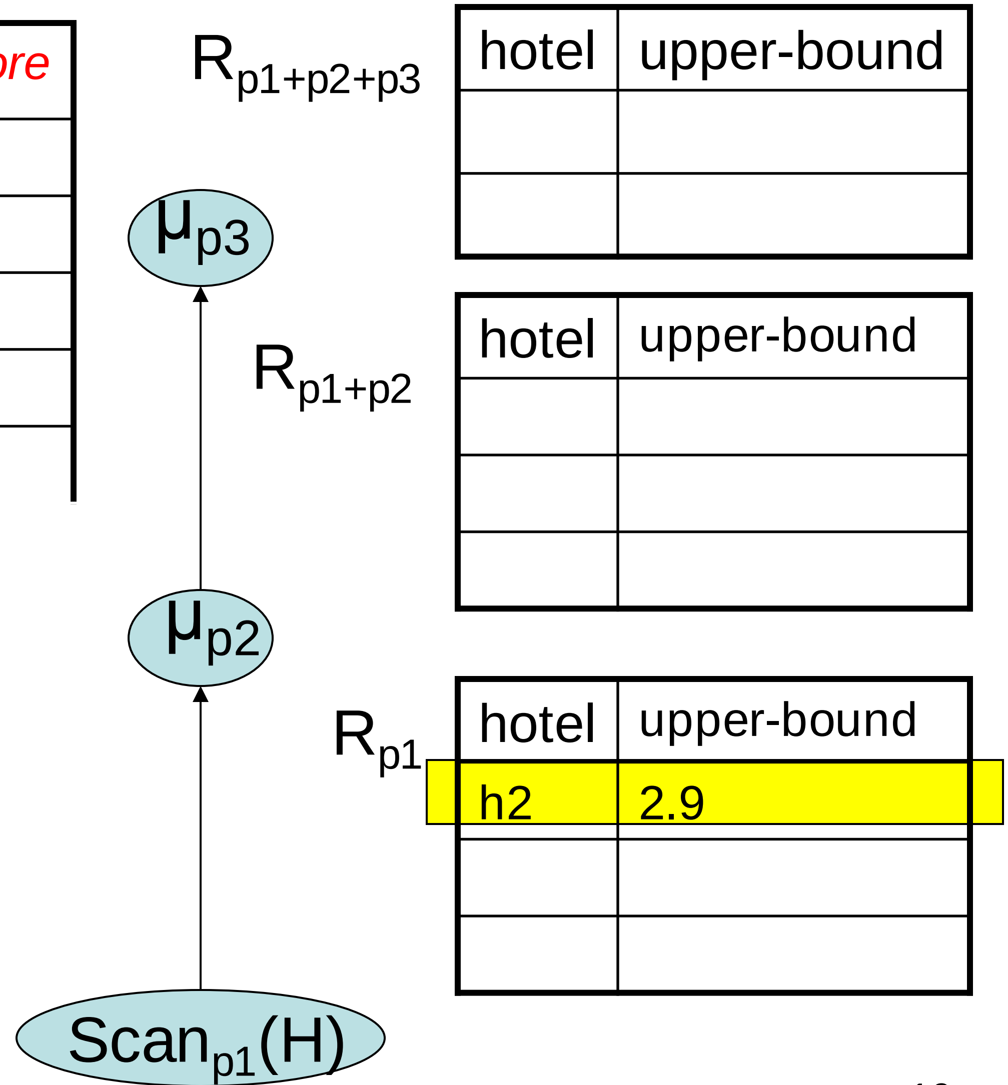
Select *
From Hotel H
Order By p1+p2+p3
Limit 1



Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.9	1.0	1.0	2.9
h2		0.9	1.0	1.0	2.9
h3		0.9	1.0	1.0	2.9
h4		0.9	1.0	1.0	2.9
…		0.9	1.0	1.0	2.9

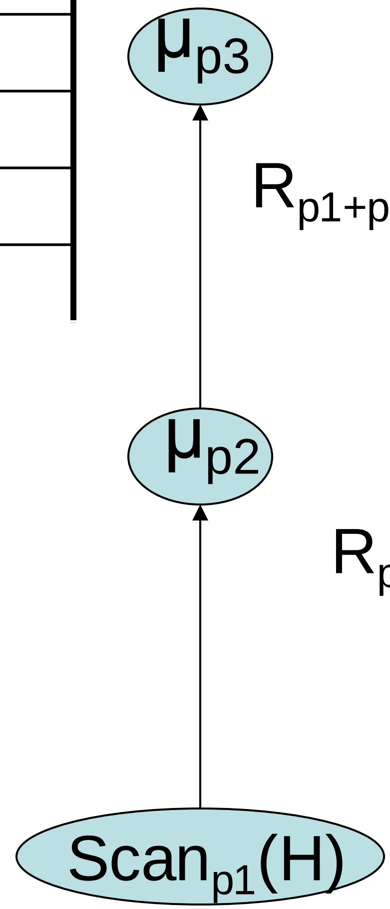
Select *
From Hotel H
Order By p1+p2+p3
Limit 1



Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.9	1.0	1.0	2.9
h2		0.9	1.0	1.0	2.9
h3		0.9	1.0	1.0	2.9
h4		0.9	1.0	1.0	2.9
…		0.9	1.0	1.0	2.9

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



$R_{p1+p2+p3}$

hotel	upper-bound

R_{p1+p2}

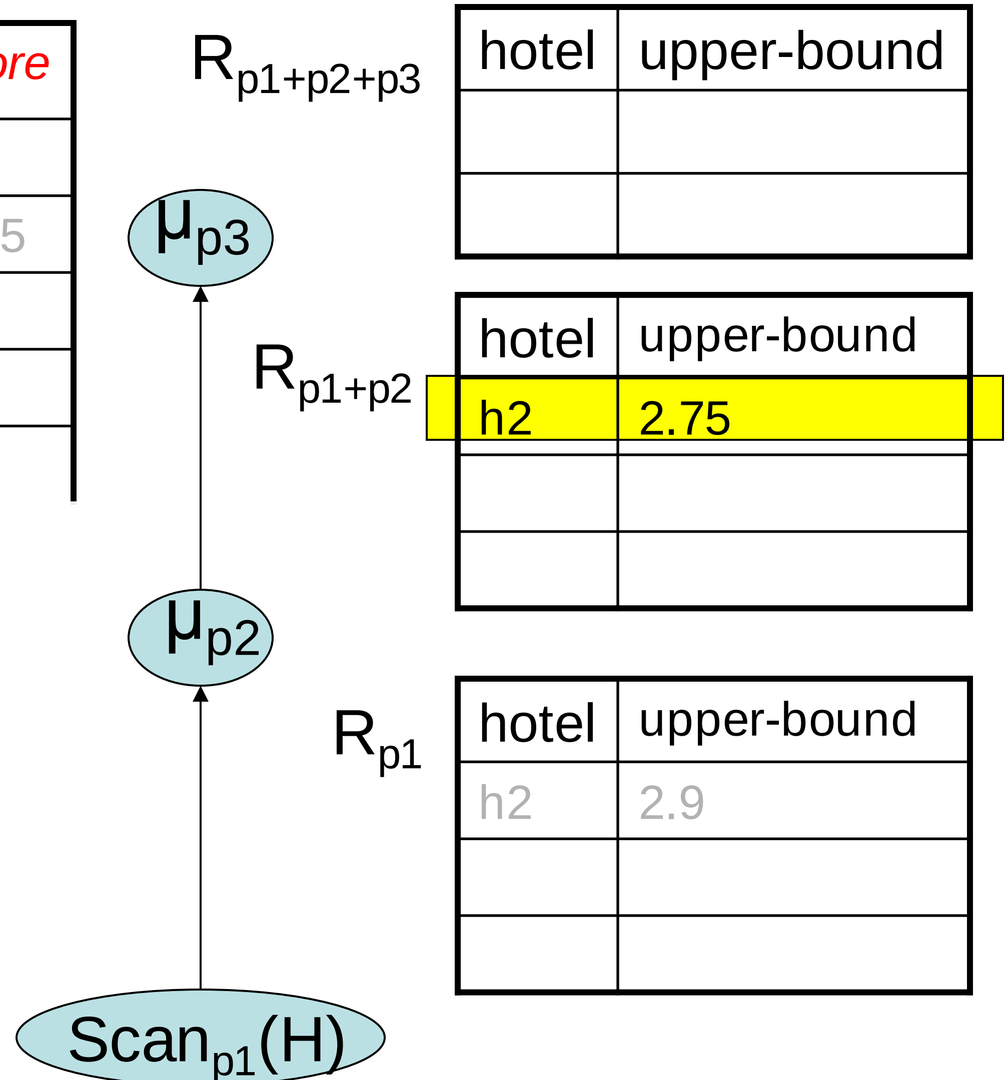
hotel	upper-bound
h2	2.9

R_{p1}

hotel	upper-bound
h2	2.9

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.9	1.0	1.0	2.9
h2		0.9	0.85	1.0	2.75
h3		0.9	1.0	1.0	2.9
h4		0.9	1.0	1.0	2.9
…		0.9	1.0	1.0	2.9

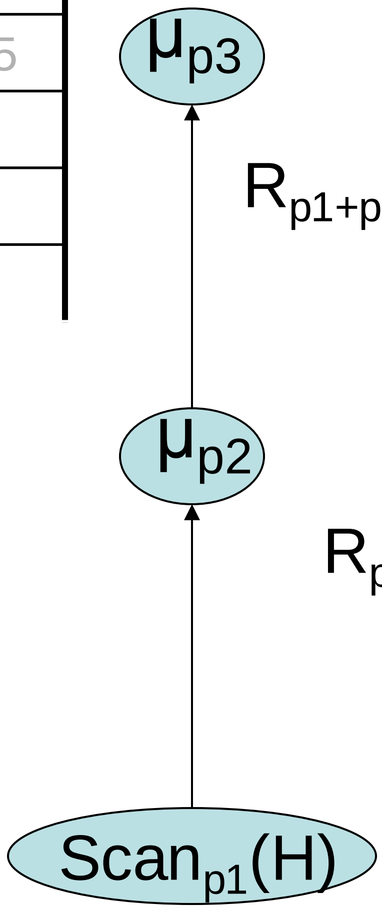


Select *
From Hotel H
Order By p1+p2+p3
Limit 1

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	1.0	1.0	2.7
h2		0.9	0.85	1.0	2.75
h3		0.7	1.0	1.0	2.7
h4		0.7	1.0	1.0	2.7
…		0.7	1.0	1.0	2.7

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



$R_{p1+p2+p3}$

hotel	upper-bound

R_{p1+p2}

hotel	upper-bound
h2	2.75

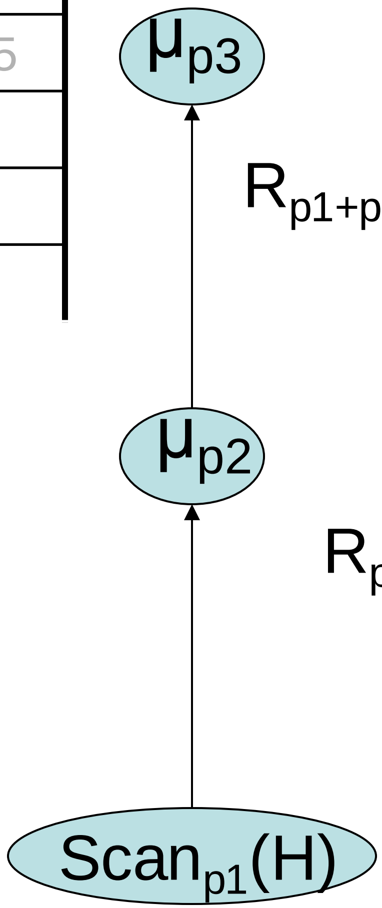
R_{p1}

hotel	upper-bound
h2	2.9
h1	2.7

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	1.0	2.5
h2		0.9	0.85	1.0	2.75
h3		0.7	1.0	1.0	2.7
h4		0.7	1.0	1.0	2.7
…		0.7	1.0	1.0	2.7

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



$R_{p1+p2+p3}$

hotel	upper-bound

R_{p1+p2}

hotel	upper-bound
h2	2.75
h1	2.5

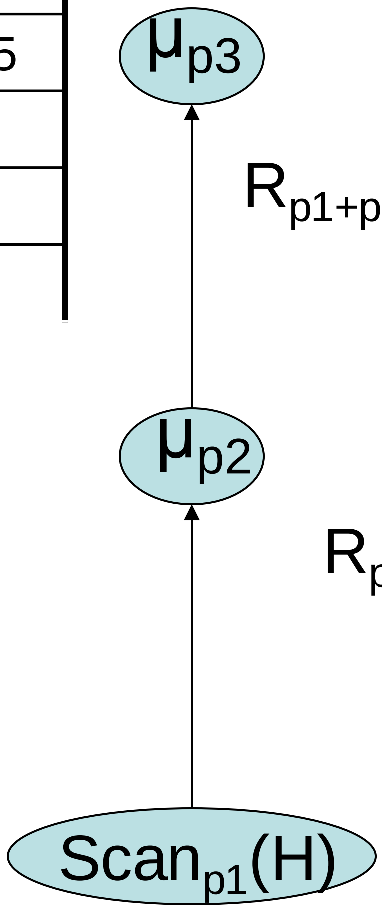
R_{p1}

hotel	upper-bound
h2	2.9
h1	2.7

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	1.0	2.5
h2		0.9	0.85	0.8	2.55
h3		0.7	1.0	1.0	2.7
h4		0.7	1.0	1.0	2.7
…		0.7	1.0	1.0	2.7

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



$R_{p1+p2+p3}$

hotel	upper-bound
h2	2.55

R_{p1+p2}

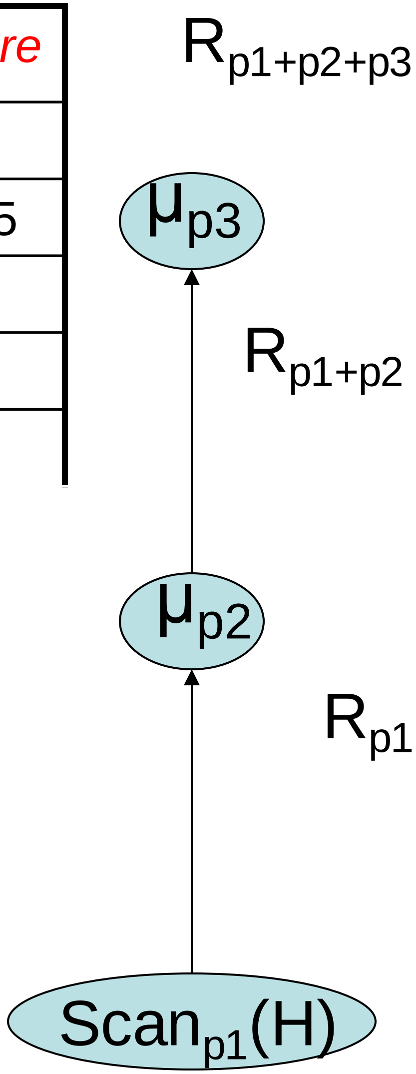
hotel	upper-bound
h2	2.75
h1	2.5

R_{p1}

hotel	upper-bound
h2	2.9
h1	2.7

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	1.0	2.5
h2		0.9	0.85	0.8	2.55
h3		0.5	1.0	1.0	2.5
h4		0.5	1.0	1.0	2.5
…		0.5	1.0	1.0	2.5



$R_{p1+p2+p3}$

hotel	upper-bound
h2	2.55

R_{p1+p2}

hotel	upper-bound
h2	2.75
h1	2.5

R_{p1}

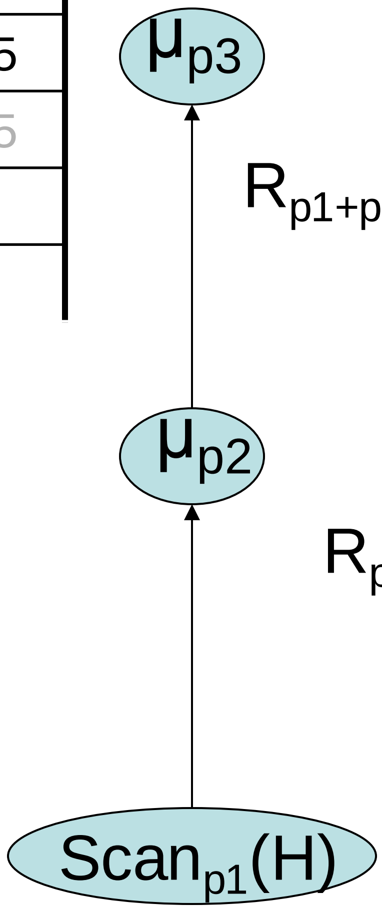
hotel	upper-bound
h2	2.9
h1	2.7
h3	2.5

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

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	1.0	2.5
h2		0.9	0.85	0.8	2.55
h3		0.5	0.45	1.0	1.95
h4		0.5	1.0	1.0	2.5
…		0.5	1.0	1.0	2.5

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



$R_{p1+p2+p3}$

hotel	upper-bound
h2	2.55

R_{p1+p2}

hotel	upper-bound
h2	2.75
h1	2.5
h3	1.95

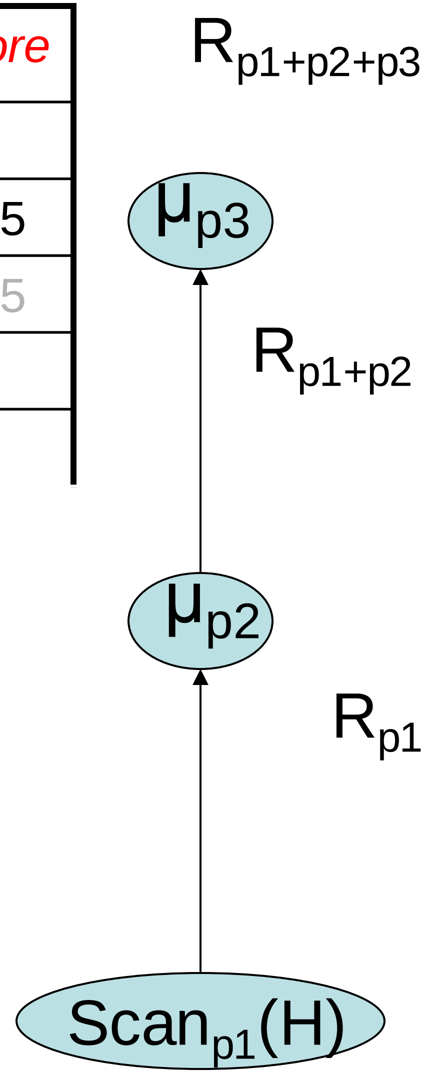
R_{p1}

hotel	upper-bound
h2	2.9
h1	2.7
h3	2.5

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	0.9	2.4
h2		0.9	0.85	0.8	2.55
h3		0.5	0.45	1.0	1.95
h4		0.5	1.0	1.0	2.5
…		0.5	1.0	1.0	2.5

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



$R_{p1+p2+p3}$:

hotel	upper-bound
h2	2.55
h1	2.4

R_{p1+p2}:

hotel	upper-bound
h2	2.75
h1	2.5
h3	1.95

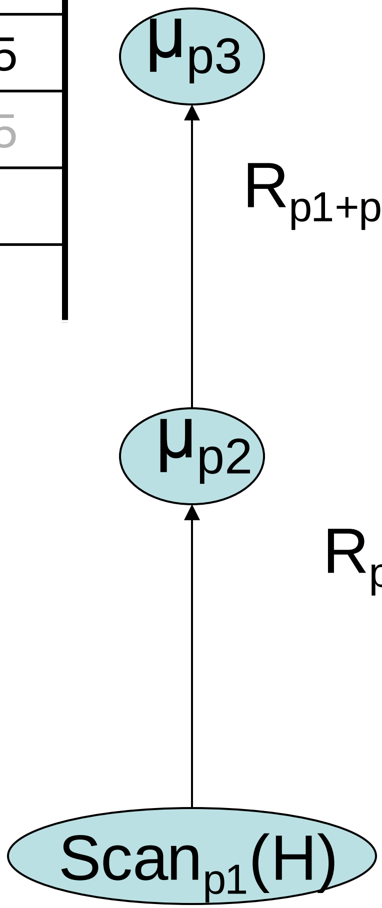
R_{p1}:

hotel	upper-bound
h2	2.9
h1	2.7
h3	2.5

Example

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	0.9	2.4
h2		0.9	0.85	0.8	2.55
h3		0.5	0.45	1.0	1.95
h4		0.5	1.0	1.0	2.5
…		0.5	1.0	1.0	2.5

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



$R_{p1+p2+p3}$

hotel	upper-bound
h2	2.55
h1	2.4

R_{p1+p2}

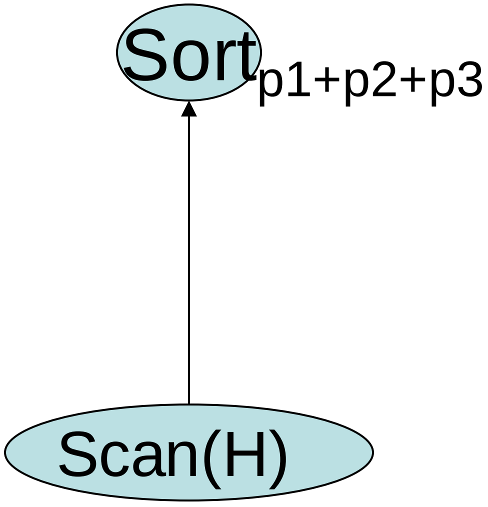
hotel	upper-bound
h2	2.75
h1	2.5
h3	1.95

R_{p1}

hotel	upper-bound
h2	2.9
h1	2.7
h3	2.5

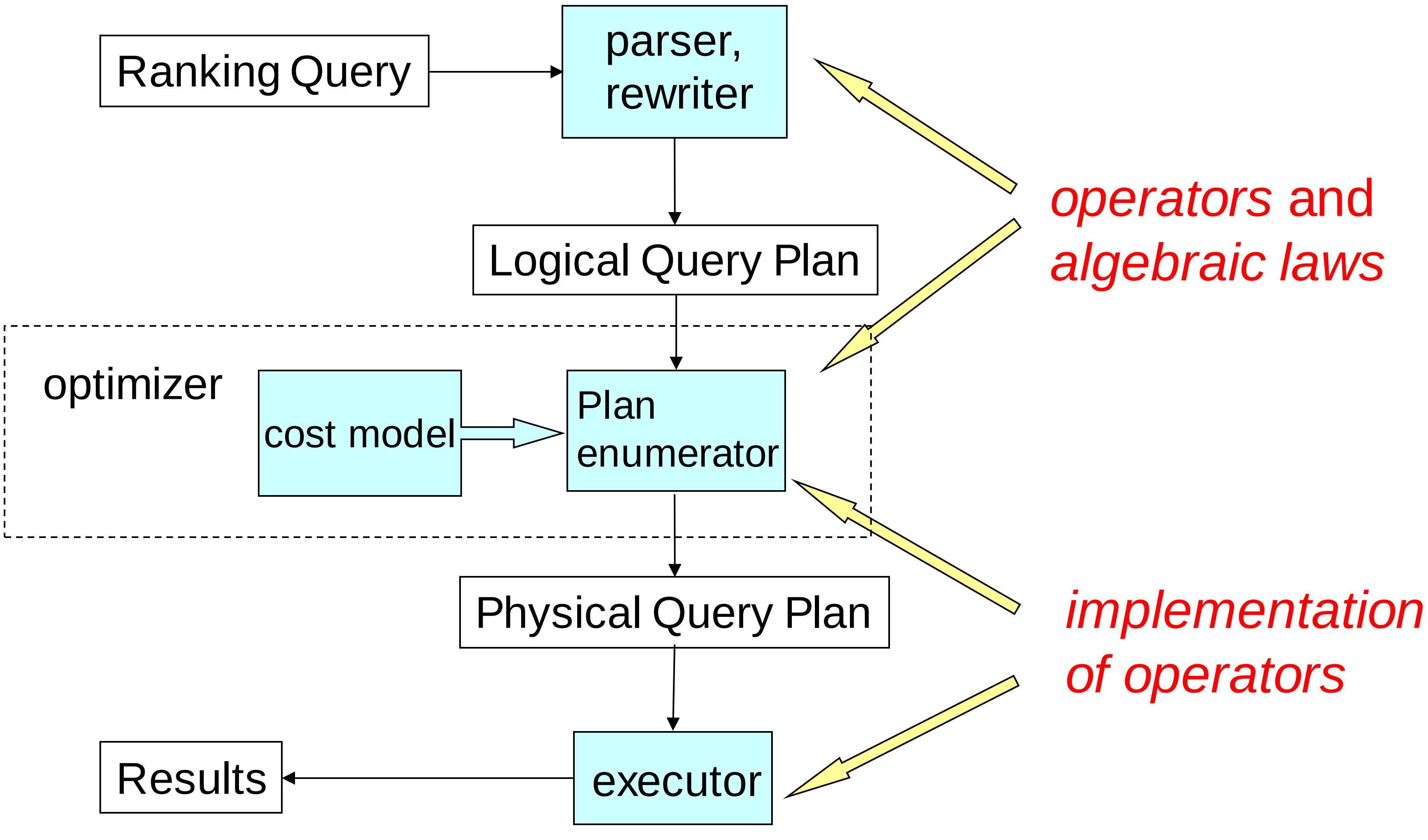
In contrast: materialize-then-sort

hotel	…	*p1*	*p2*	*p3*	*score*
h1		0.7	0.8	0.9	2.4
h2		0.9	0.85	0.8	2.55
h3		0.5	0.45	0.75	1.7
h4		0.4	0.7	0.95	2.05
…		…	…	…	….



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

Impact of Rank-Relational Algebra



Optimization

- Two-dimensional enumeration:

  ranking (ranking predicate scheduling)

  and

  filtering (join order selection)

- Sampling-based cardinality estimation

Two-Dimensional Enumeration

- (1 table, 0 predicate)
  seqScan(H), idxScan(H), seqScan(M), …
- (1 table, 1 predicate)
  $rankScan_{cheap}$(H), μ_{cheap}(seqScan(H)), …
- (1 table, 2 predicates)
  μ_{close}($rankScan_{cheap}$(H)), …
- (2 table, 0 predicate)
  NestLoop(seqScan(H), seqScan(M)), …
- (2 table, 1 predicate)
  NRJN($rankScan_{cheap}$(H), seqScan(M)),…
- 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!