RankSQL: Supporting Ranking Queries in RDBMS

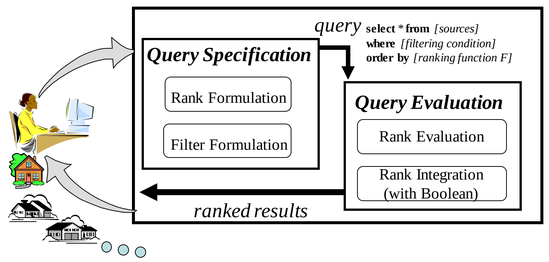
Chengkai Li, Mohamed A. Soliman, Kevin Chen-Chuan Chang, Ihab F. Ilyas

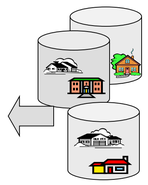
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN, UNIVERSITY OF WATERLOO

The Context: *AIMing to the top*

The ultimate goal of the AIM project --

Supporting ranking in data retrieval





Relational DBMS or middleware sources (e.g,, Web)

The Problem: *Supporting Ranking in RDBMS*

- Support *ranking* as a first-class query type
- Integrate *ranking* with Boolean query constructs

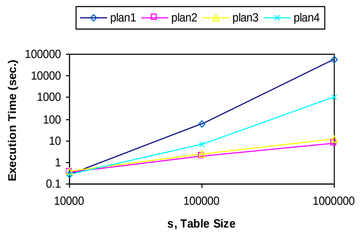
HID	MID	cheap	close	related	score
h1	m2	0.9	0.7	0.8	2.4
h2	m1	0.6	0.8	0.9	2.3
h1	m3	0.9	0.7	0.6	2.2

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

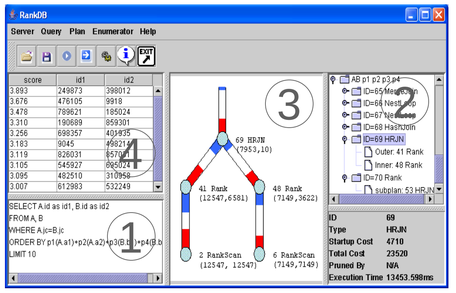
$\mathcal{B}$ *membership dimension: Boolean predicates, Boolean function*

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

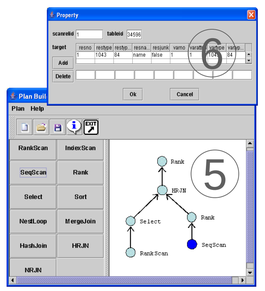
The Demonstration:



Orders of magnitude cost difference:
plan 1: traditional plan
plan 2-4: new ranking query plans



RankSQL front-end: (1) query; (2) enumerator visualizer; (3) execution monitor; (4) results.

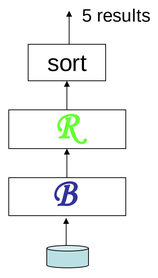


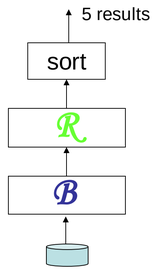
Plan Builder: (5) plan editor; (6) operator property.

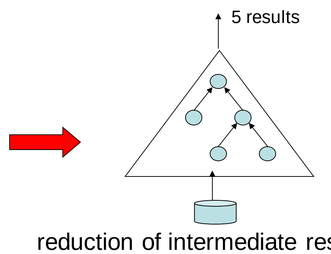
Our Challenges:

RDBMS treats ranking as second class:
- Monolithic ranking component $\mathcal{R}$,
- Processed after Boolean component $\mathcal{B}$.

Naïve ***materialize-then-sort*** scheme:
- Only 5 top results are requested, whole results are scored and ordered;
- Scan and join unnecessary tuples;
- Ranking predicates can be expensive: *cheap(h.price)*: online source *related(m.collection,"dinosaur")*:IR *close(h.addr,"O'Hare airport")*: querying geographical data







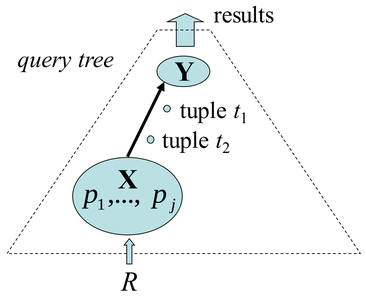
reduction of intermediate results, thus processing cost

Our Insight: *Splitting and Interleaving*

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

Our Principle: *Ranking Principle*

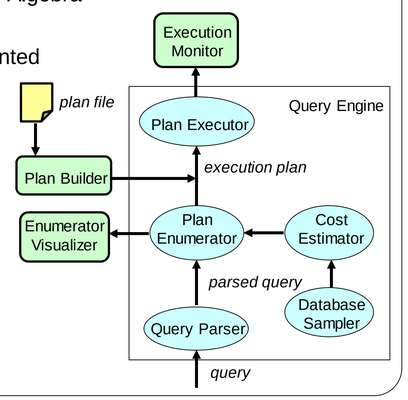
Tuples should be processed in the order of their **upper-bound scores** with respect to the evaluated predicates.



Our Solution: *RankSQL [SIGMOD'05]*

Foundation: Rank-Relational Algebra
- Data model: rank-relation
- Operators: new and augmented
- Algebraic laws

Impact: Query Engine
- Executor:
  - Physical operators implementation
  - Incremental plans
- Optimizer:
  - Plan enumeration
  - Cost estimation



Task 1: *Algebraic Foundation*

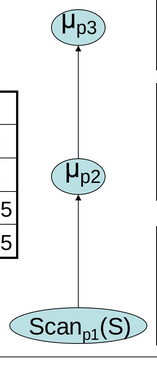
Rank-Relational Algebra
- Data Model: Rank-Relation S_P

  S (membership); P (order) : evaluated predicates
- Operators:
  - Order dimension: μ (rank) operator

    Satisfying splitting requirement;
  - Membership dimension: augmented rank-aware operators σ π ∩ ∪ −▷◁

    Satisfying interleaving requirement.
- Algebraic laws: e.g., $\sigma(\mu(S_P)) \equiv \mu(\sigma(S_P))$

TID	p1	p2	p3
s1	0.7	0.8	0.9
s2	0.9	0.85	0.8
s3	0.5	0.45	0.75
s4	0.4	0.7	0.95



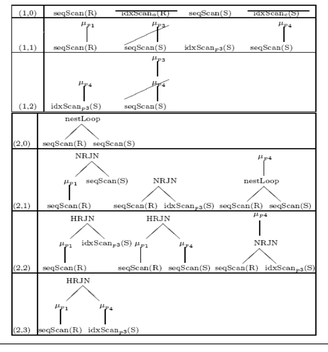
TID	upper-bound
s2	2.55
s1	2.4

TID	upper-bound
s2	2.75
s1	2.5
s3	1.95

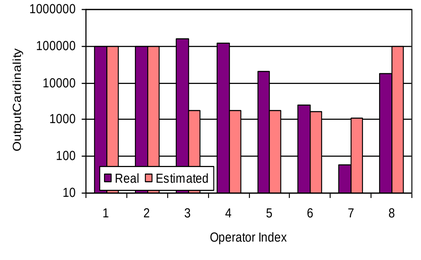
TID	upper-bound
s2	2.9
s1	2.7
s3	2.5

Task 2: *Rank-Aware Query Optimizer*

Two-Dimensional Plan Enumeration:



Sampling-Based Cardinality Estimation:



Contributions: *Summary*

For supporting ranking in RDBMS, we developed:
- **Key insight:**
  - Splitting and interleaving
- **Fundamental Principle:**
  - Ranking principle
- **Algebraic foundation:**
  - Rank-Relation
  - New and augmented operators
  - Algebraic laws
- **Optimization Framework:**
  - Two-dimensional plan enumeration
  - Sampling-based cardinality estimation

DAIS **The Database and Information Systems Laboratory**
at The University of Illinois at Urbana-Champaign
Large Scale Information Management