Reverse Engineering Top-k Join Queries

Kiril Panev
panev@cs.uni-kl.de

Nico Weisenauer
n_weisenau10@cs.uni-kl.de

Sebastian Michel
smichel@cs.uni-kl.de

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Top-3 Customers

Bruce Campbell	1000
John Doe	749.90
Adam Miller	199.99


```
SELECT c.name, max(o.price)
FROM    customers c,
        orders o
WHERE c.customer_id = o.customer_id
AND     c.country = 'England'
GROUP BY c.name
ORDER by max(o.price) DESC
LIMIT 3
```

Why and Where?

```
SELECT c.name, max(o.price)
FROM    customers c,
        orders o
WHERE c.customer_id = o.customer_id
AND     c.country = 'England'
GROUP BY c.name
ORDER by max(o.price) DESC
LIMIT 5
```

- Alternative queries
  - Different joins
- List from other source
  - A customized result can be produced by modifying the query
- Finding explanatory SQL Queries
  - E.g., for crowd-sourced top-k rankings
- Related Work
  - E.g., [Zhang et al., SIGMOD '13], [Psalidas et al., SIGMOD '15]
  - Do not handle top-k queries with aggregations

Customer ID	Name	Country	Balance
1	John Doe	England	250.49
2	Adam Miller	England	124.56
7	Sam Burns	Scotland	154.67
12	Benjamin Smith	Wales	1955.22
50	Bruce Campbell	England	45.99
…	…	…	…

Customers

Order ID	Customer ID	Price	Date
1	1	24.50	11/28/14
2	1	749.90	04/01/15
23	2	22.49	12/01/11
24	2	199.99	12/30/12
78	50	1.99	10/01/12
79	50	1000.00	02/27/15
…	…	…	…

Orders

entity	score
Bruce Campbell	1000.00
John Doe	749.90
Adam Miller	199.99

L

entity

score?

Customer ID	Name	Country	Balance
1	John Doe	England	250.49
2	Adam Miller	England	124.56
7	Sam Burns	Scotland	154.67
12	Benjamin Smith	Wales	1955.22
50	Bruce Campbell	England	45.99
…	…	…	…

Customers

score?

Order ID	Customer ID	Price	Date
1	1	24.50	11/28/14
2	1	749.90	04/01/15
23	2	22.49	12/01/11
24	2	199.99	12/30/12
78	50	1.99	10/01/12
79	50	1000.00	02/27/15
…	…	…	…

Orders

entity	score
Bruce Campbell	1000.00
John Doe	749.90
Adam Miller	199.99

L

entity

Customer ID	Name	Country	Balance
1	John Doe	England	250.49
2	Adam Miller	England	124.56
7	Sam Burns	Scotland	154.67
12	Benjamin Smith	Wales	1955.22
50	Bruce Campbell	England	45.99
…	…	…	…

Customers

score

Order ID	Customer ID	Price	Date
1	1	24.50	11/28/14
2	1	749.90	04/01/15
23	2	22.49	12/01/11
24	2	199.99	12/30/12
78	50	1.99	10/01/12
79	50	1000.00	02/27/15
…	…	…	…

Orders

entity	score
Bruce Campbell	1000.00
John Doe	749.90
Adam Miller	199.99

L

```
SELECT c.name, max(o.price)
FROM    customers c,
         orders o
WHERE c.customer_id = o.customer_id
AND     c.country = 'England'
GROUP BY c.name
ORDER by max(o.price) DESC
LIMIT 5
```

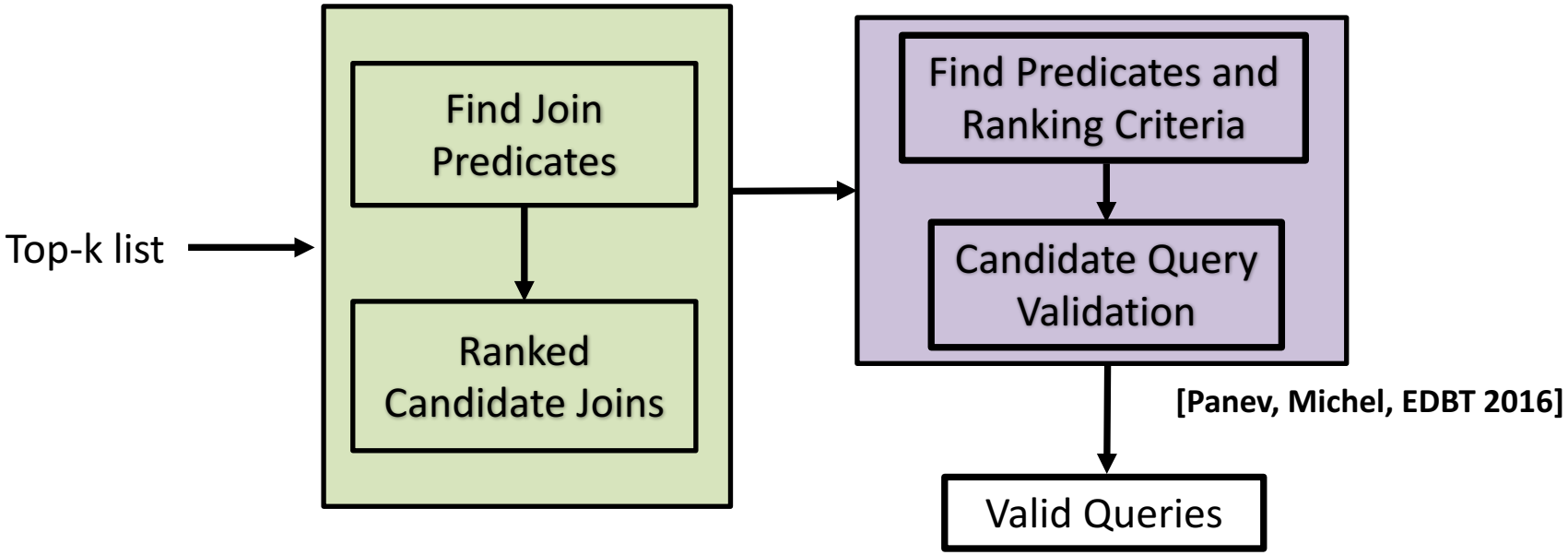
Challenges

- Given a **database D** and an **input list L**

  We want to find Q such that $Q(D) = L$

- Minimum database interaction
  - Avoid query execution
- Identify queries that would be the best candidates in producing L

The PALEO-J Framework



entity	score
Bruce Campbell	1000.00
John Doe	749.90
Adam Miller	199.99

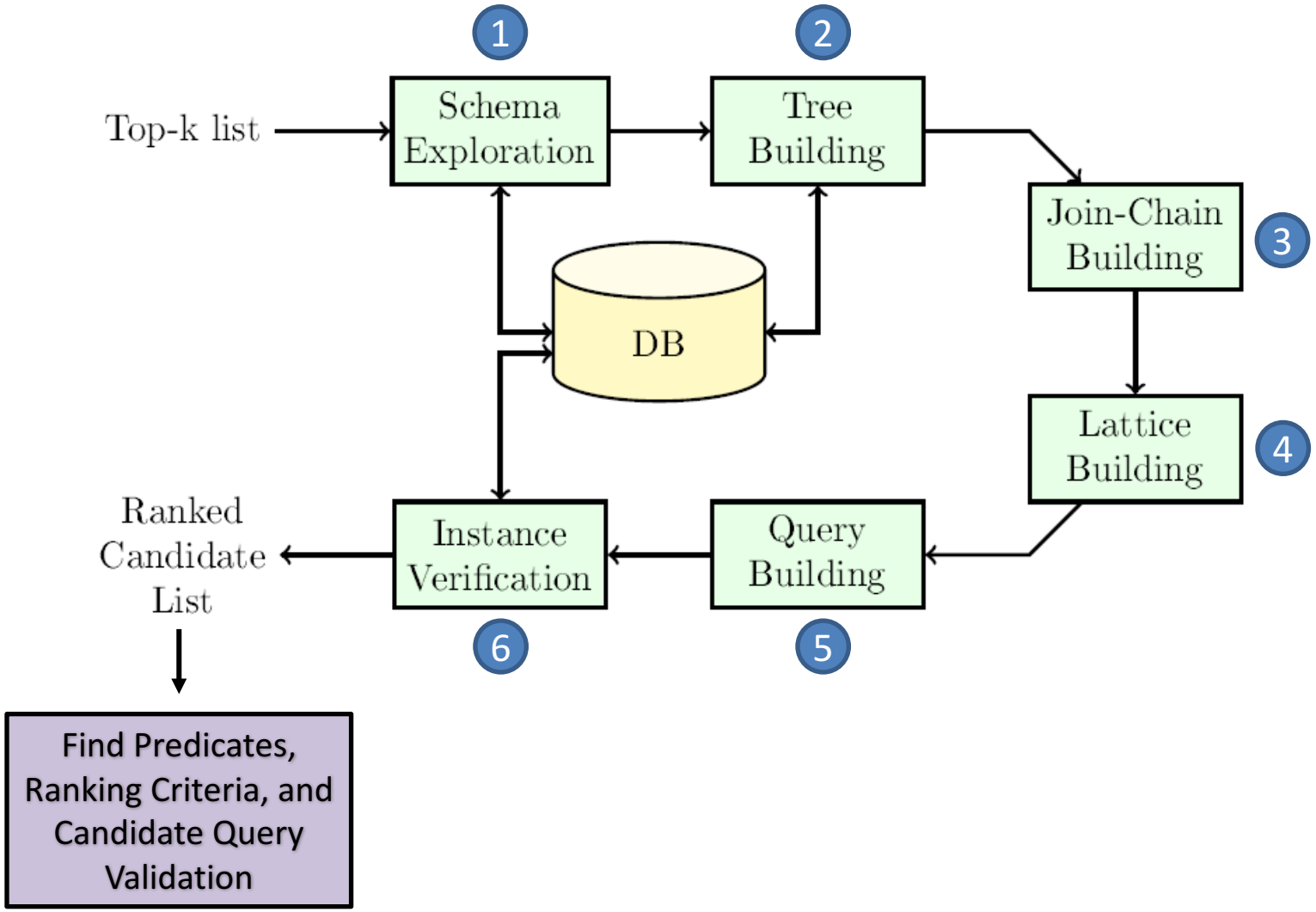
join predicate

SELECT entity, score
FROM A, B, …
WHERE A.id = B.a_id …
AND P_1 **and** P_2 **and** …
GROUP BY entity
ORDER BY 2 **DESC**
LIMIT k

Outline

- Introduction
  - Problem Statement
- PALEO-J
  - Finding Join Predicates
- Optimizations
- Experimental Evaluation
- Conclusion

Finding Join Predicates



Step 1: Schema Exploration

entity score?

Customer ID	Name	Country	Balance
1	John Doe	England	250.49
2	Adam Miller	England	124.56
7	Sam Burns	Scotland	154.67
12	Benjamin Smith	Wales	1955.22
50	Bruce Campbell	England	45.99
…	…	…	…

Customers

score?

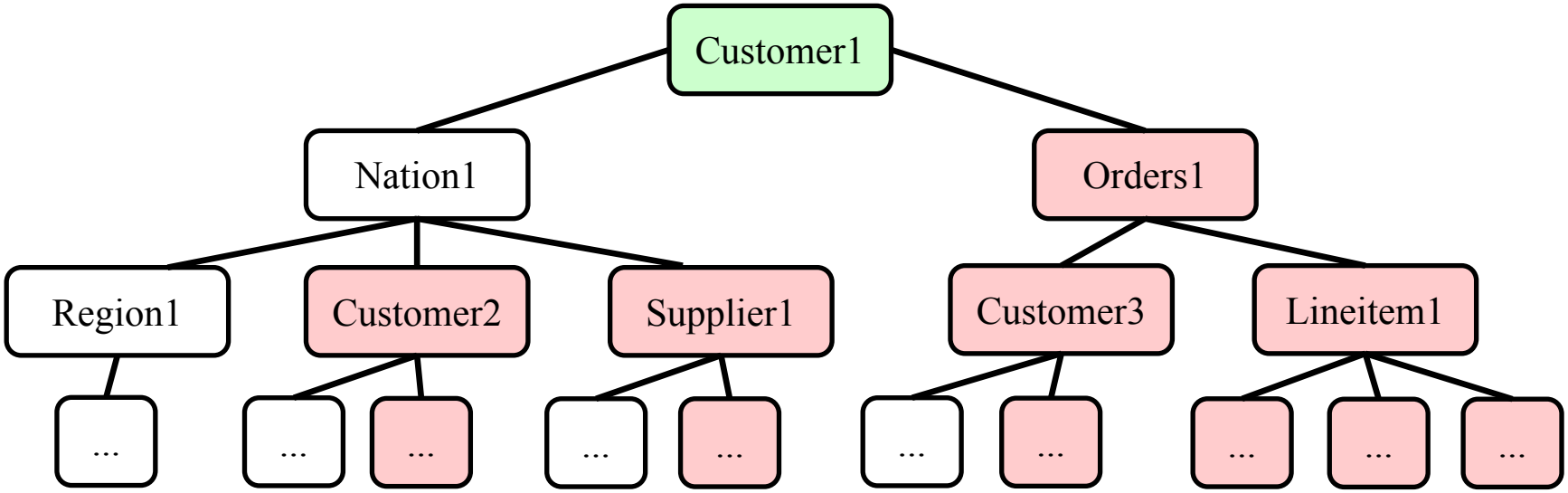
Order ID	Customer ID	Price	Date
1	1	24.50	11/28/14
2	1	749.90	04/01/15
23	2	22.49	12/01/11
24	2	199.99	12/30/12
78	50	1.99	10/01/12
79	50	1000.00	02/27/15
…	…	…	…

Orders

L

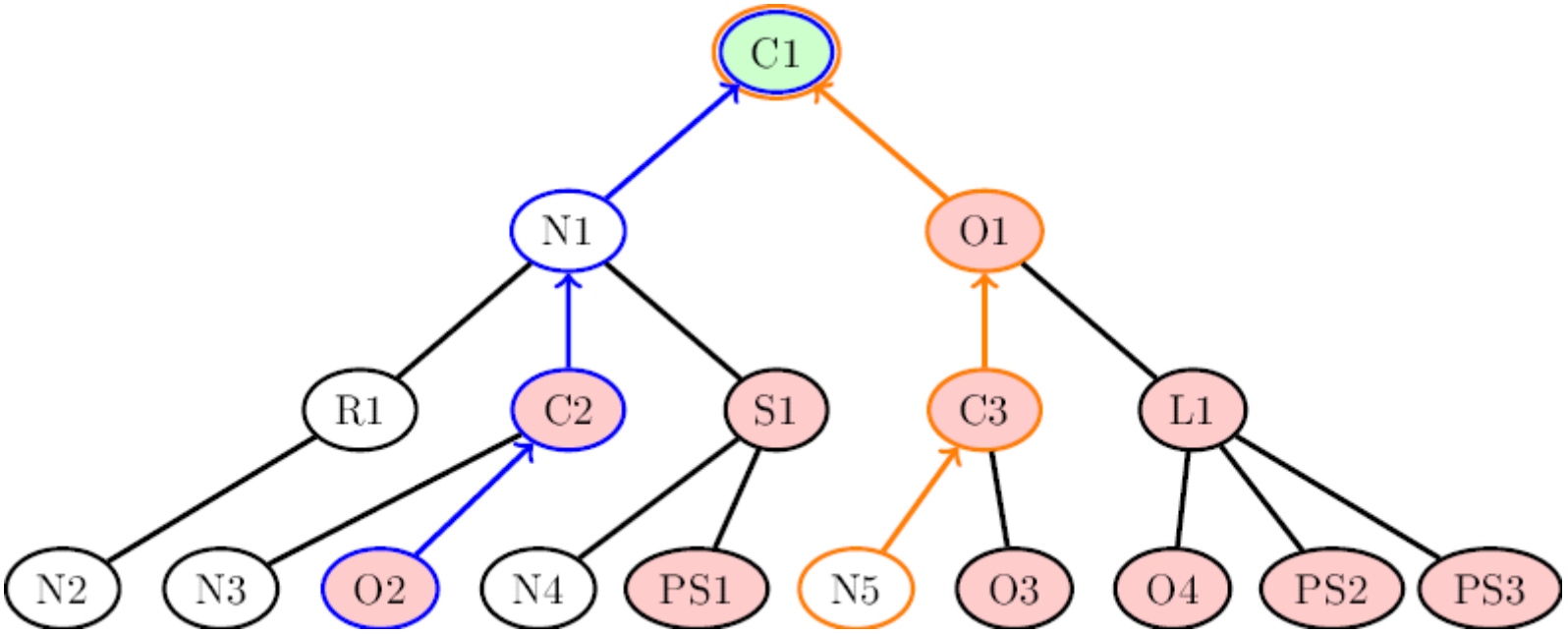
entity	score
Bruce Campbell	1000.00
John Doe	749.90
Adam Miller	199.99

Step 2: Tree Building



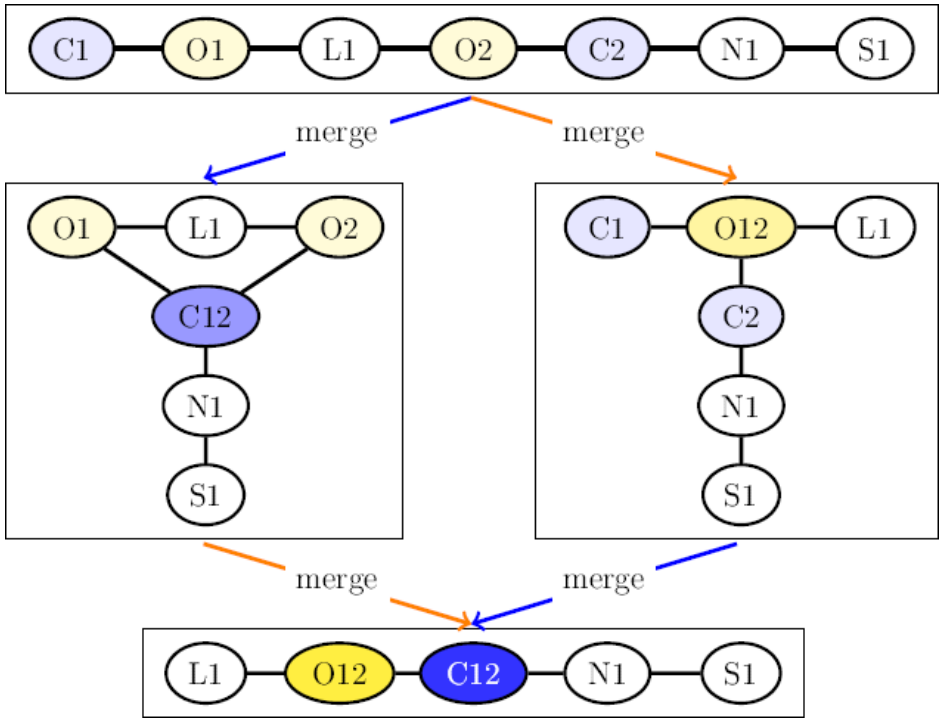
- Build a tree by following key constraints, starting from the table containing the entity column to a pre-defined depth d
- Tables containing score columns are marked red

Step 3: Join Chain Building



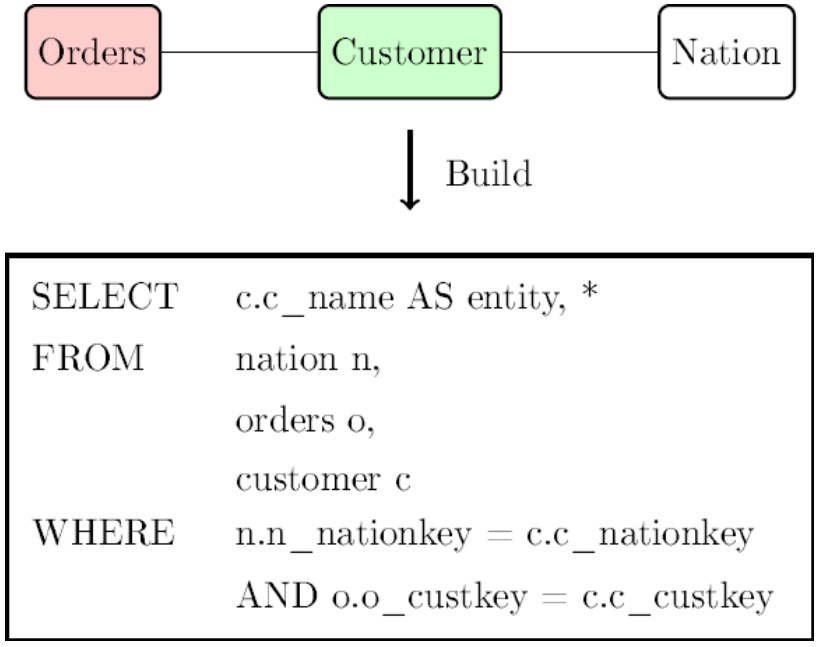
- A join chain has to contain the table with the entity column and at least one table with a score column

Step 4: Node Merging



- Merge nodes that correspond to the same database tables
- Each merge step generates a new query graph

Step 5: Query Building



- Build the SQL query statement from the query graphs
- Calculate the cost of queries by estimating join result sizes
- Rank the join candidates by cost

Step 6: Instance Verification

Input List

entity	score
e1	100
e2	90
e3	80

Candidate Join Chain

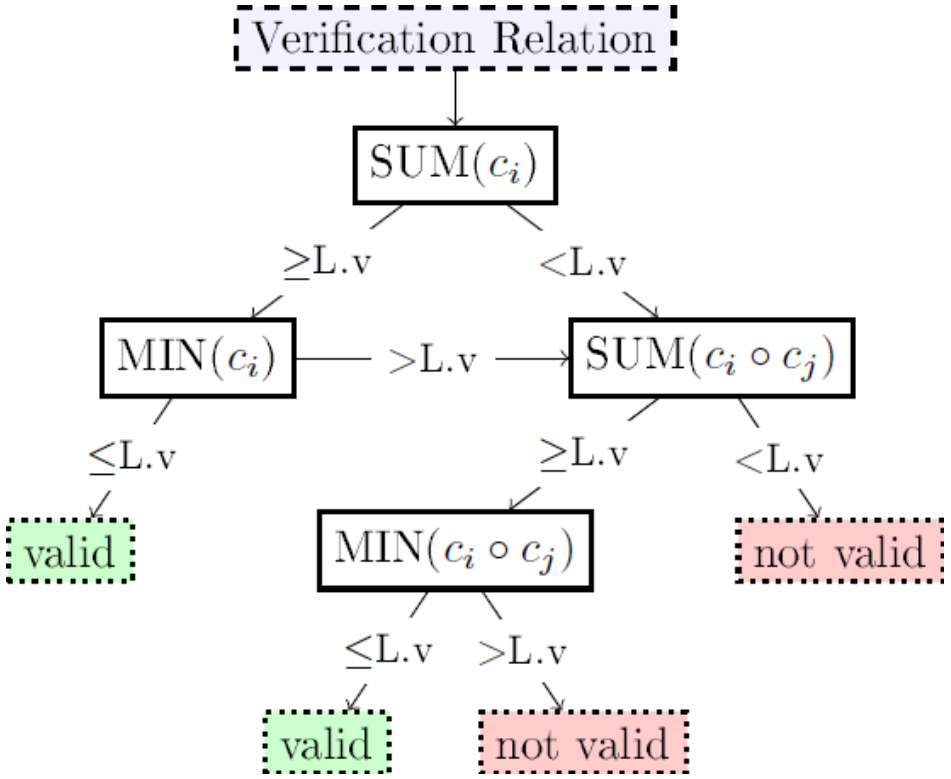

```
SELECT   A.entity, *
FROM     A, B, C
WHERE    A.id = B.id
AND      B.id = C.id
AND      A.entity = 'e1'
```

Verification Relation

A.entity	...	B.score	...	C.score	...
e1	...	105	...	0.2	...
e1	...	95	...	1.2	...
e1	...	100	...	0.1	...

SUM(B.score)	MIN(B.score)	SUM(C.score)	MIN(C.score)
300	95	1.5	0.1

Decision Tree for Validity of a Candidate Join



$\circ \in \{+, \times\}$

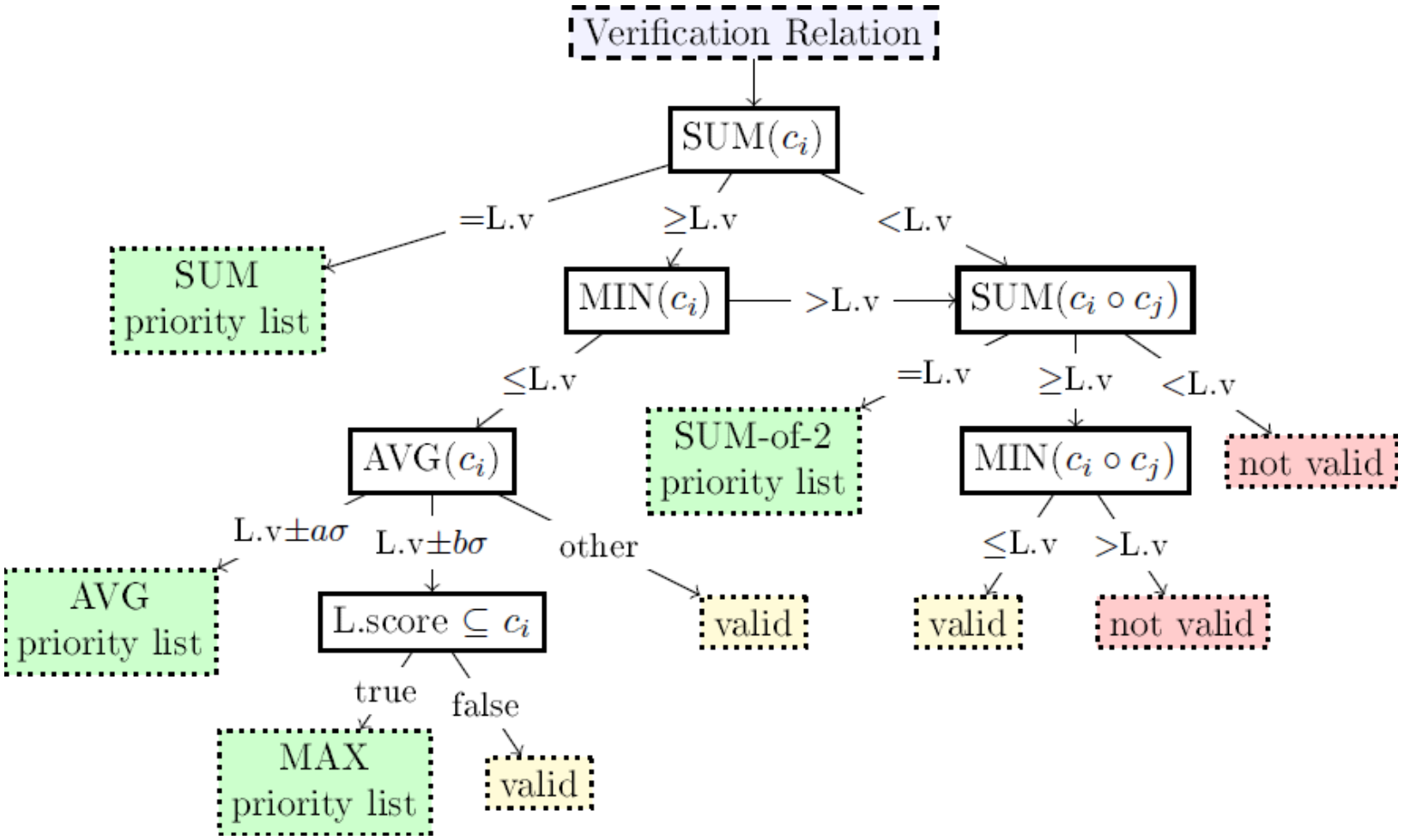
L.v
100

SUM(B.score)	MIN(B.score)	SUM(C.score)	MIN(C.score)
300	95	1.5	0.1

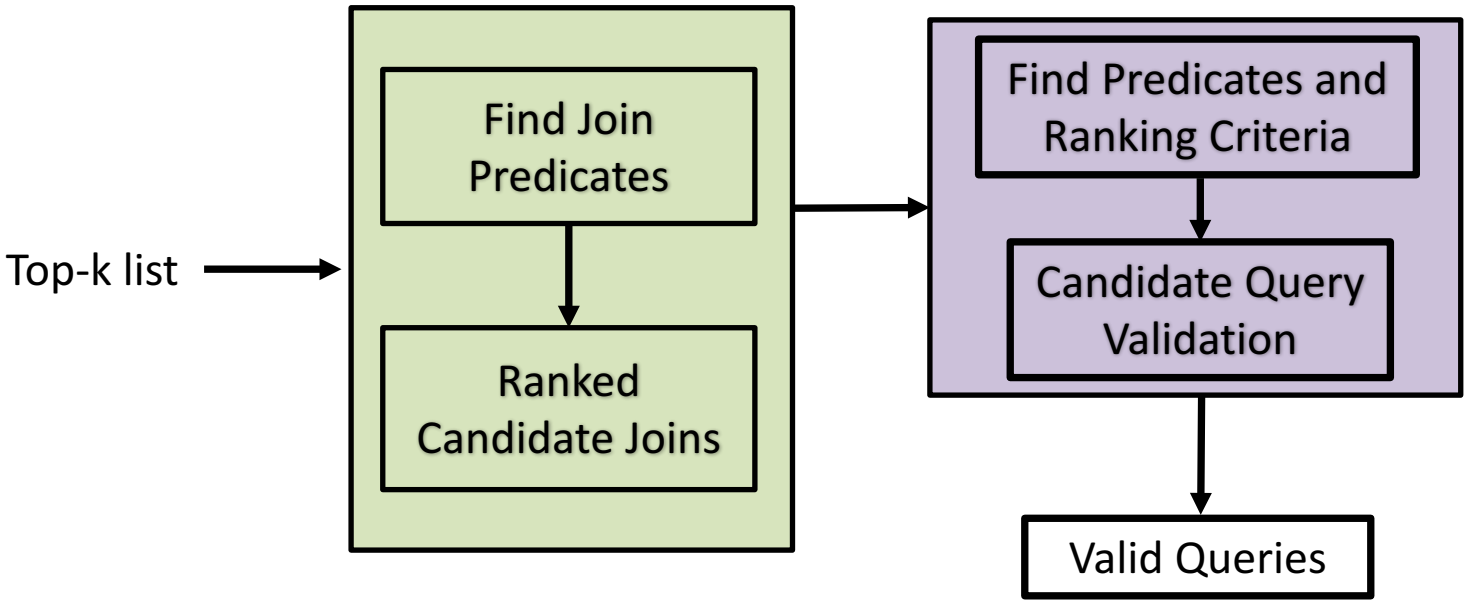
Advanced Optimization

- AVG priority list:
  - Check if the score comes close to the mean value of a column
- MAX priority list:
  - Check columns for inclusion of score values
- SUM/SUM-of-2 priority list:
  - Check if the sum of a column matches the score
- Candidate joins in priority lists will be checked first!

Advanced Decision Tree



Query Discovery with PALEO-J



Outline

- Introduction
  - Problem Statement
- PALEO-J
  - Finding Join Predicates
- Optimizations
- Experimental Evaluation
- Conclusion

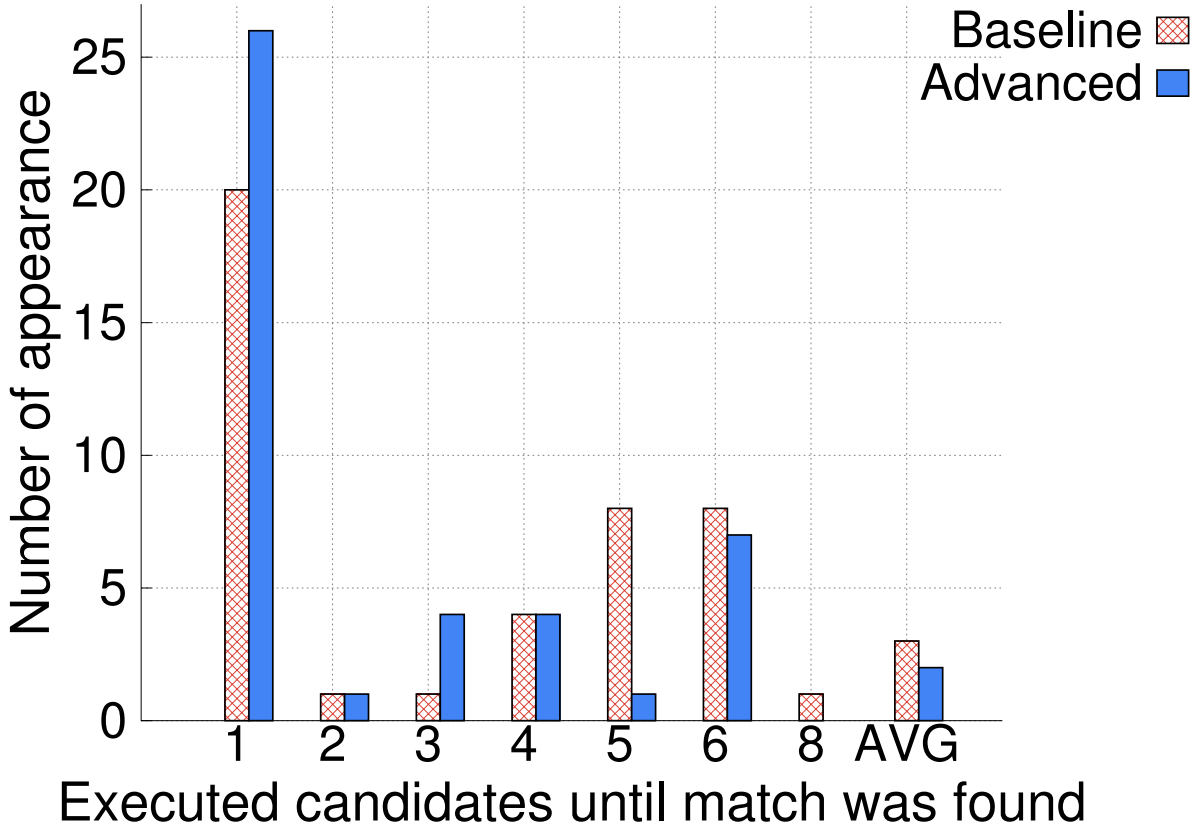
Experimental Evaluation

- TPC-H Dataset 10GB
- Workloads
  - 43 Queries based on TPC-H
    - with 1-3 joins
    - adjusted to create supported query types

      score: **max**(A), **avg**(A), **sum**(A), **sum**(A+B), **sum**(A∗B), **no-agg**
- Maximum depth d is set to 5

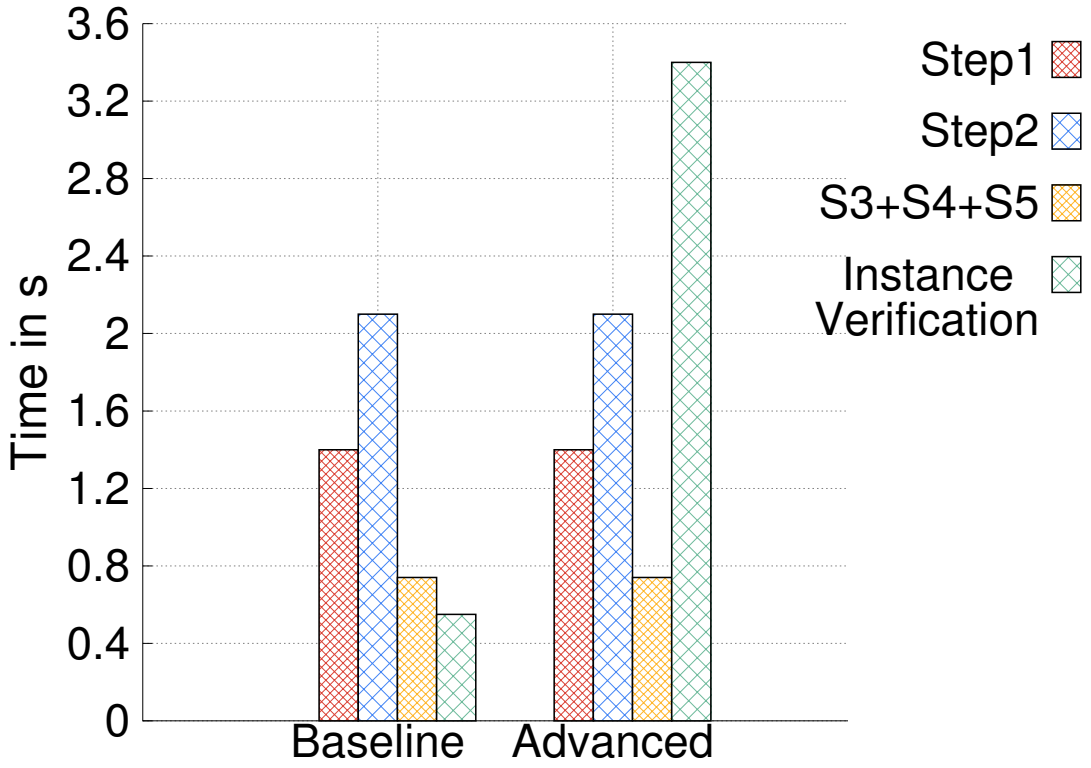
We find all of the queries!

Number of candidate joins examined until a valid query is found



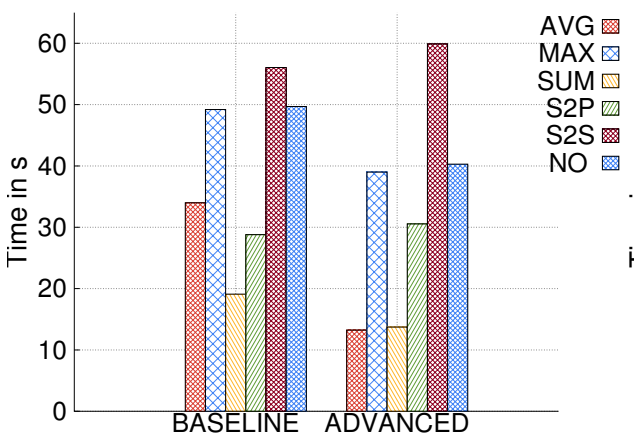
Most of the queries are found by inspecting the first candidate join!

Average runtime of the different steps in finding the join predicate

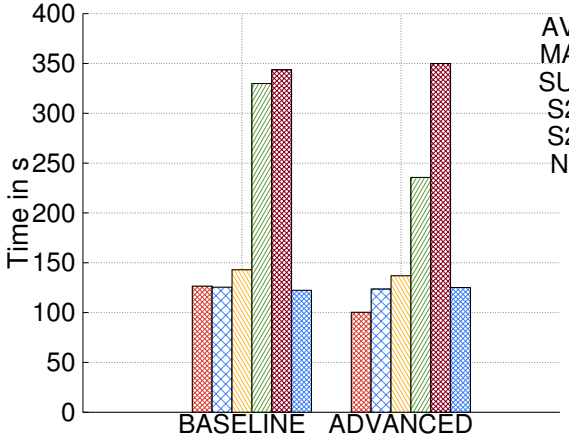


The overhead in the advanced approach a small cost to pay compared to the benefit in the next steps

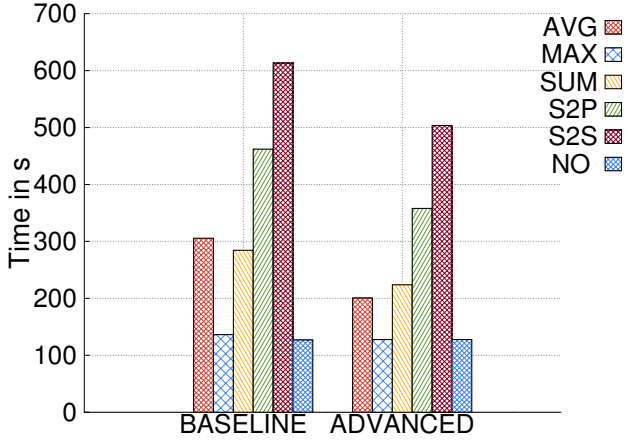
Average runtime for finding a valid query by different query graph size



(a) 1-Join



(b) 2-Joins



(c) 3-Joins

TPC-H

The advanced approach outperforms the baseline

Conclusion

- PALEO-J reverse engineers top-k join queries
  - Find join predicates
    - 6-step approach
- Advanced approach
  - Overhead in Instance Verification improves candidate join ranking
- Always discovers a valid query
  - Average of 2 candidate join examinations with the advanced approach

Use-Case: Exploring Databases

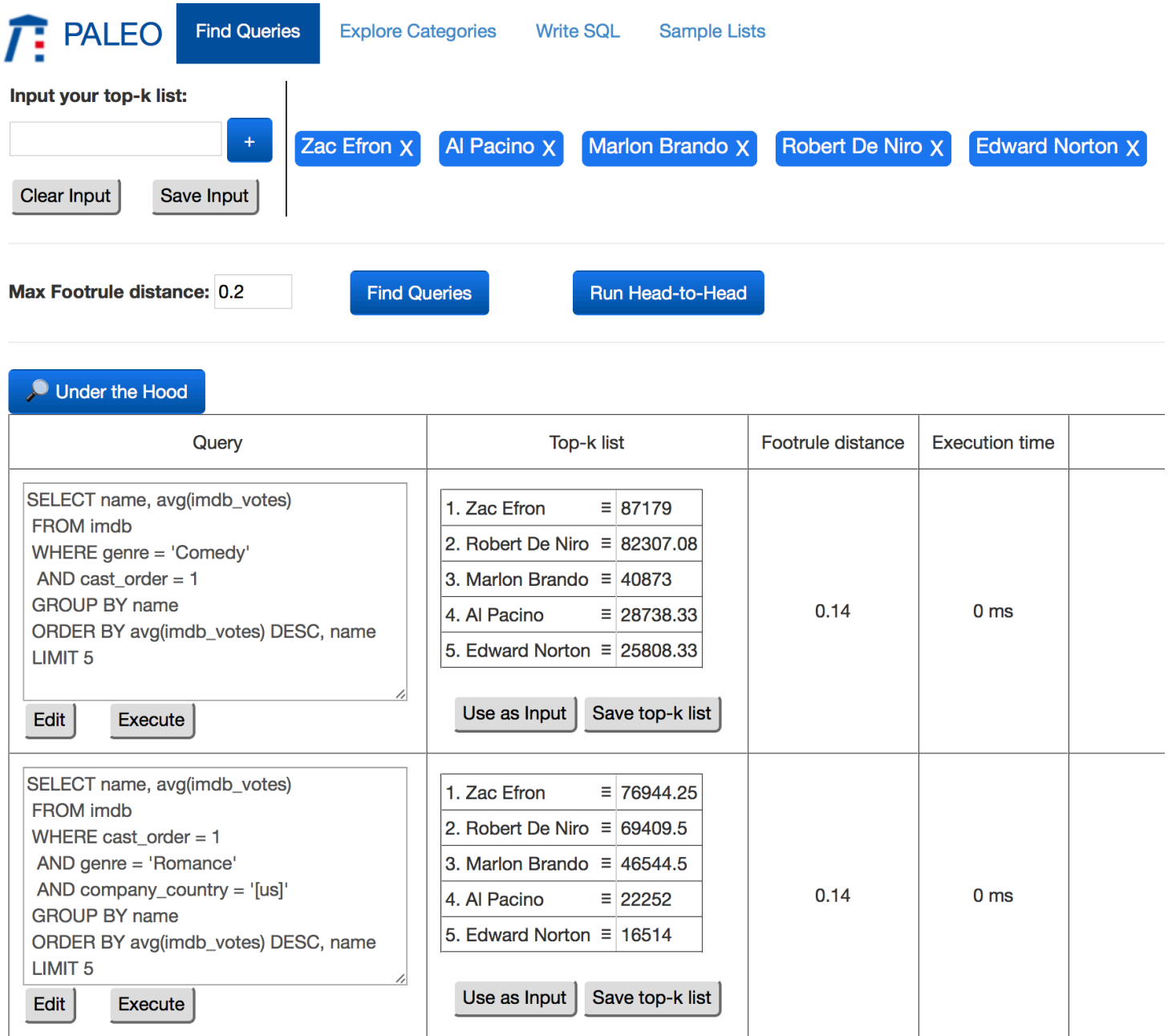


Image References

- [1] http://cdn2.hubspot.net/hubfs/51294/Product_Site/Images/Engineering_-_Hover.png?t=1453269935144
- [2] http://gounconventional.com/files/2011/11/tf2_engineer_by_cutekakashi.jpg
- [3] https://www.asme.org/getmedia/8fec1f0b-f060-4fc9-92b8-f22844957835/Engineering-and-Business-A-Combination-for-Success_hero.jpg.aspx