

Data-analysis and Retrieval

Top-k searching

Hans Philippi

May 10, 2023

Top-k queries

An example top-k query:

```
SELECT * FROM UsedCars  
WHERE brand = 'BMW' OR brand = 'Mercedes'  
ORDER BY 8*price + 2*mileage  
LIMIT 20
```

General characteristics:

We order resulting tuples according to some function f and are interested in a limit set of tuples containing extreme values of f .

Another example: similarity matching in image databases.

Given a certain image qim:

```
SELECT im.id FROM Images im
WHERE date >= '01.01.2016'
ORDER BY DESC
      0.5*ColorSim(qim, im) + 0.5*TextureSim(qim, im)
LIMIT 10
```

Top-k query processing

A naive approach to top-k query processing:

- Process the basic structure of the query (before the ORDER)
- Do the sorting based on the function f
- Scan the first k tuples from the resulting table

So in general, finding a top-10 might require processing 10,000 or 10^6 or 10^9 tuples.

Can we do better?

Top-k query processing

A naive approach to top-k query processing:

- Process the basic structure of the query (before the ORDER)
- Do the sorting based on the function f
- Scan the first k tuples from the resulting table

So in general, finding a top-10 might require processing 10,000 or 10^6 or 10^9 tuples.

Can we do better?

Yes, we can!

(But there are some conditions)

Top-k query processing: monotonicity

- We require that f is monotone
- In our examples, we suppose f is *monotonic non-decreasing*¹
- Intuition: if you increase the value one of the parameters of f and keep the others constant, f will not decrease.
- Definition: suppose f has n parameters.
For each $i \in [1..n]$ we require that for all x_1, x_2, \dots, x_n and y_i :
$$x_i < y_i \implies f(x_1, x_2, \dots, x_i, \dots, x_n) \leq f(x_1, x_2, \dots, y_i, \dots, x_n)$$

¹Of course, with some adaptations, we could also apply the algorithms if f is monotonic non-increasing.

Top-k algorithms

- We will look at two algorithms by Fagin e.a.
- We suppose that we access the data columns through *lists*
- TA: Threshold Algorithm
sorted access and random access to each list
- NRA: No Random Access algorithm
sorted access to each list, but no random access

Top-k: lists representing function parameters

We represent the attributes which are arguments of f by separate lists. Remember that we suppose f has to be maximized and is monotonic non-decreasing. The values of lists A and B each have a known minimum (in our case 0).

Example: the table below has sorted access to both A and B.

<i>oid</i>	A	B	C
001	4	7	...
002	2	8	...
003	3	1	...

For TA and NRA, it is represented as:

<i>oid</i>	A
001	4
003	3
002	2

<i>oid</i>	B
002	8
001	7
003	1

Top-k: lists in a relational context

- If we have an RDBMS with a classical, record oriented architecture, we can realize sorted access by creating a B-tree index on each column that acts as a list.
- Caveat: note that you cannot manipulate the B-tree directly. You are forced to simulate the manipulation of the B-tree using SQL-queries.
- If we have a main-memory RDBMS with column stores, sorted access can be obtained by efficient internal sorting techniques.
- Random access based on *oid* can be realized either by an index structure or by a (duplicate) list sorted on oid.

TA: Threshold Algorithm

Basic approach: in each round of the algorithm, one new value from each list is inspected, starting at the top. Suppose $f = A + B$.

<i>oid</i>	A
001	4
003	3
002	2

<i>oid</i>	B
002	8
001	7
003	1

We keep track of the following variables:

Max-A: maximum value of A to be found in following rounds (4)

Max-B: maximum value of B to be found in following rounds (8)

Threshold T : maximum value of f to be found in next rounds (12)

Buffer: known $[oid, f]$ tuples: $[001, 11]$, $[002, 10]$

Partial top-k: still empty (why?)

TA: Threshold Algorithm

Intermezzo 1: TA
(four rounds)

TA: Threshold Algorithm

OID	A
4	100
1	90
6	80
5	70
2	40
3	30

OID	B
6	100
5	90
1	70
3	50
4	30
2	20

ROUND	1	2	3	4
Max-A	100			
Max_B	100			
Treshold	200			
Buffer	[4:130] [6:180]			
Top-k				

TA: Threshold Algorithm

OID	A
4	100
1	90
6	80
5	70
2	40
3	30

OID	B
6	100
5	90
1	70
3	50
4	30
2	20

ROUND	1	2	3	4
Max-A	100	90		
Max_B	100	90		
Treshold	200	180		
Buffer	[4:130] [6:180]	[4:130] *[6:180] [1:160] [5:160]		
Top-k		[6:180]		

NRA: No random access algorithm

Basic approach: in each round of the algorithm, one new value from each list is inspected, starting at the top, but now we cannot find the corresponding oid in the other list quickly.

<i>oid</i>	A	<i>oid</i>	B
001	4	002	8
003	3	001	7
002	2	003	1

We keep track of the following variables:

Max-A: maximum value of A to be found in next rounds (4)

Max-B: maximum value of B to be found in next rounds (8)

Threshold T : maximum value of f to be found in next rounds (12)

Buffer: known [oid, f] tuples: [001, 4 - 12], [002, 8 - 12]

Partial top-k: still empty (why?)

NRA: No random access algorithm

Intermezzo 2: NRA
(four rounds)

NRA: No Random Access Algorithm

OID	A
4	90
1	90
6	40
5	30
2	20
3	10

OID	B
6	100
5	40
1	40
4	30
3	20
2	10

ROUND	1	2	3	4
Max-A	90			
Max_B	100			
Treshold	190			
Buffer (changing!)	[4: 90 - 190] [6: 100 - 190]			
Top-k				

NRA: No Random Access Algorithm

OID	A
4	90
1	90
6	40
5	30
2	20
3	10

OID	B
6	100
5	40
1	40
4	30
3	20
2	10

ROUND	1	2	3	4
Max-A	90	90		
Max_B	100	40		
Treshold	190	130		
Buffer (changing!)	[4: 90 - 190] [6: 100 - 190]	[4: 90 - 130] [6: 100 - 190] [1: 90 - 130] [5: 40 - 130]		
Top-k				

Top-k: more advanced approaches

We have seen top-k query processing on single tables. More complications rise when the query involves a join of several tables.

The reference below describes an approach where top-k processing is fully integrated with classical relational query optimization.

https:

[//cs.uwaterloo.ca/~ilyas/papers/ilyassigmod05.pdf](https://cs.uwaterloo.ca/~ilyas/papers/ilyassigmod05.pdf)

- In 2013, Nick Schuiling and Maarten van Duren implemented these algorithms on top of MonetDB, a main-memory column store RDBMS developed at the CWI. They realized huge performance gains on TPC/H benchmarks.
- You are invited to apply smart top-k algorithms in Lab 1.