Updated 1/28/2022

Topic 1: Data models and query languages
Unit 1: SQL (continued)
Lecture 4

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

1/28/2022

Pre-class conversations

- Last class recapitulation
  - including outer joins as unions, matrix chain multiplication, full disjunction
- Any more questions on class procedures?
  - Hybrid: leave feedback after class today if you have suggestions
  - When submitting your first scribes, please also send me an email

- today:
  - SQL continued (with connections to optimization and algebra)
  - Logic

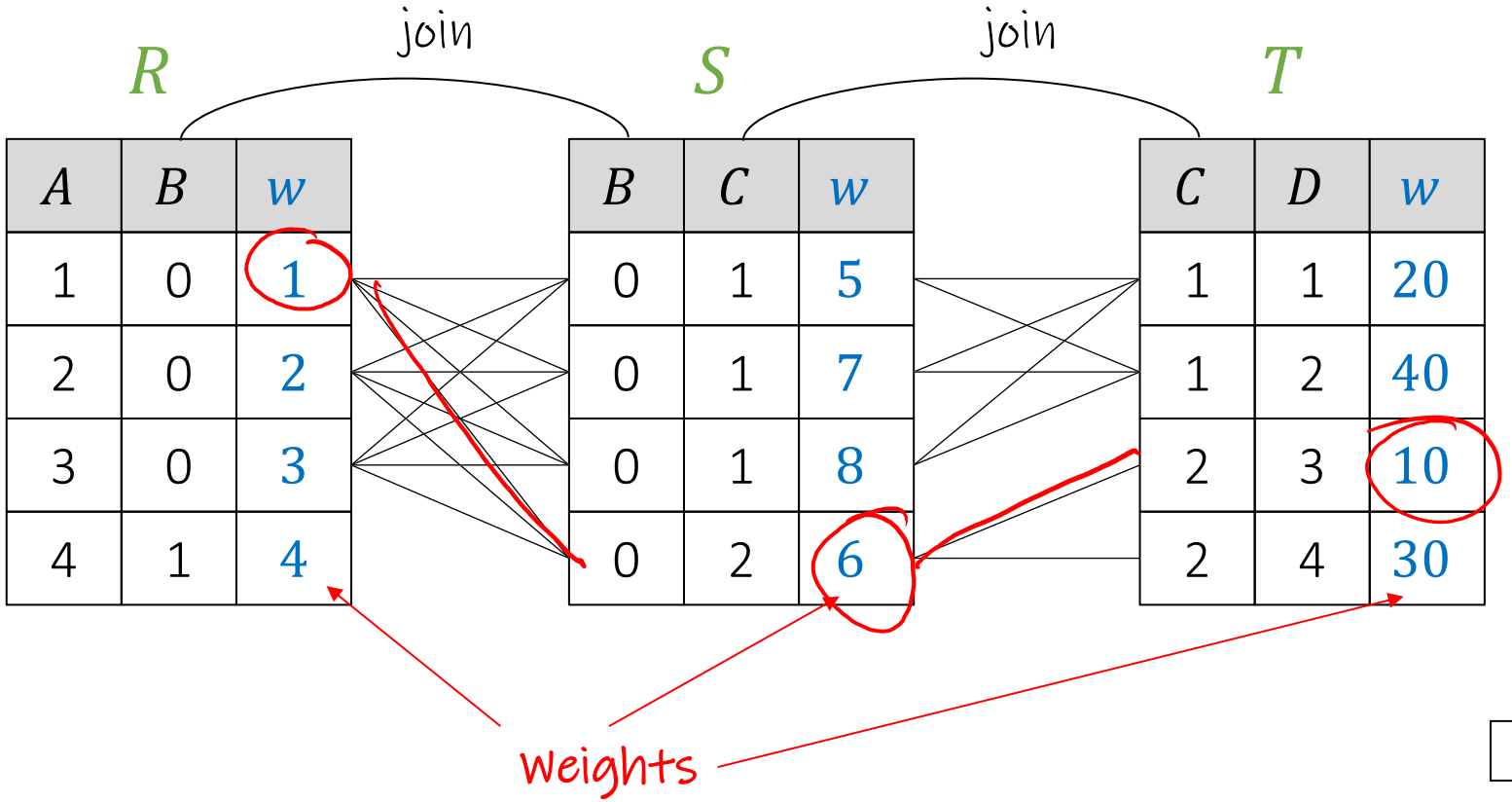
PRELIMINARY

- **Lecture 1 (Tue 1/18):** Course introduction, SQL, PostgreSQL setup, SQL Activities
- **Lecture 2 (Fri 1/21):** SQL
- **Lecture 3 (Tue 1/25):** SQL
- **Lecture 4 (Fri 1/28):** Logic & Relational Calculus
- **Lecture 5 (Tue 2/1):** Relational Algebra & Codd's Theorem
- **Lecture 6 (Fri 2/4):** Relational Algebra & Codd's Theorem
- **Lecture 7 (Tue 2/8):** Datalog & Recursion,
- **Lecture 8 (Fri 2/11):** Datalog & Recursion,
- **Lecture 9 (Tue 2/15):** Alternative Data Models

Pointers to relevant concepts & supplementary material:

- **Unit 1. SQL**: [SAMS'12], [CS 3200], [Cow'03] Ch3 & Ch5, [Complete'08] Ch6, [Silberschatz+'20] Ch3.8
- **Unit 2. Logic & Relational Calculus**: First-Order Logic (FOL), relational calculus (RC): [Barland+'08] 4.1.2 & 4.2.1 & 4.4, [Genesereth+] Ch6, [Halpern+'01], [Cow'03] Ch4.3 & 4.4, [Elmasri, Navathe'15] Ch8.6 & Ch8.7, [Silberschatz+'20] Ch27.1 & Ch27.2, [Alice'95] Ch3.1-3.3 & Ch4.2 & Ch4.4 & Ch5.3-5.4, [Barker-Plummer+'11] Ch11
- **Unit 3. Relational Algebra & Codd's Theorem**: Relational Algebra (RA), Codd's theorem: [Cow'03] Ch4.2, [Complete'08] Ch2.4 & Ch5.1-5.2, [Elmasri, Navathe'15] Ch8, [Silberschatz+'20] Ch2.6, [Alice'95] Ch4.4 & Ch5.4
- **Unit 4. Datalog & Recursion**: Datalog, recursion, Stratified Datalog with negation, Datalog evaluation strategies, Stable Model semantics, Answer Set Programming (ASP): [Complete'08] Ch5.3, [Cow'03] Ch 24, [Koutris'19] L9 & L10, [G., Suciu'10]
- **Unit 5. Alternative Data Models**: NoSQL: [Hellerstein, Stonebraker'05], [Sadalage, Fowler'12], [Harrison'16]

Sorting & Top-*k* evaluation with SQL



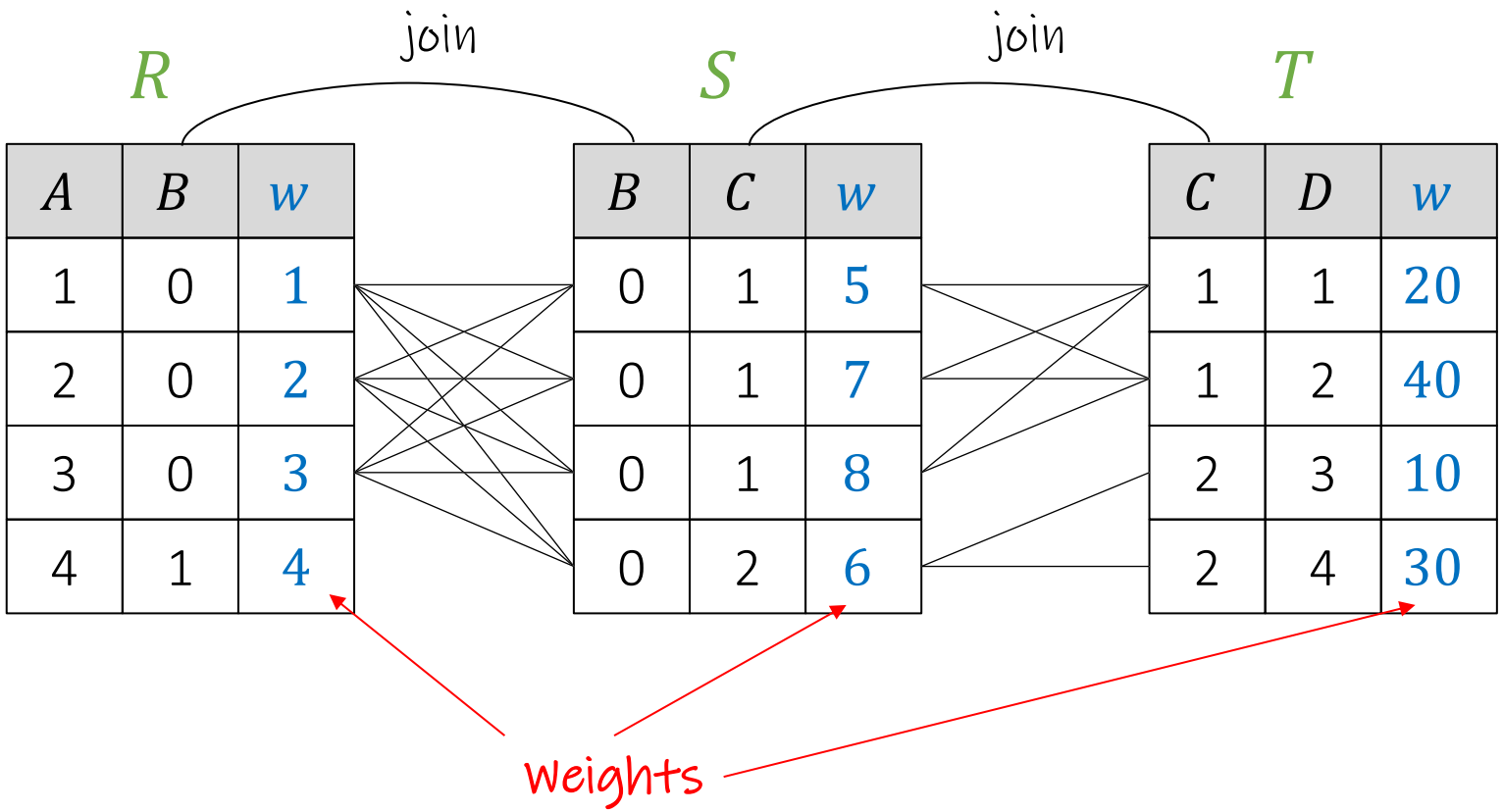

```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
```

Return all 24 results in order of sum of weights

Result

A	B	C	D	weight
1	0	2	3	17
2	0	2	3	18
3	0	2	3	19
1	0	1	1	26
2	0	1	1	27
3	0	1	1	28
1	0	1	1	28
...	...	...	...	...

Sorting & Top-*k* evaluation with SQL



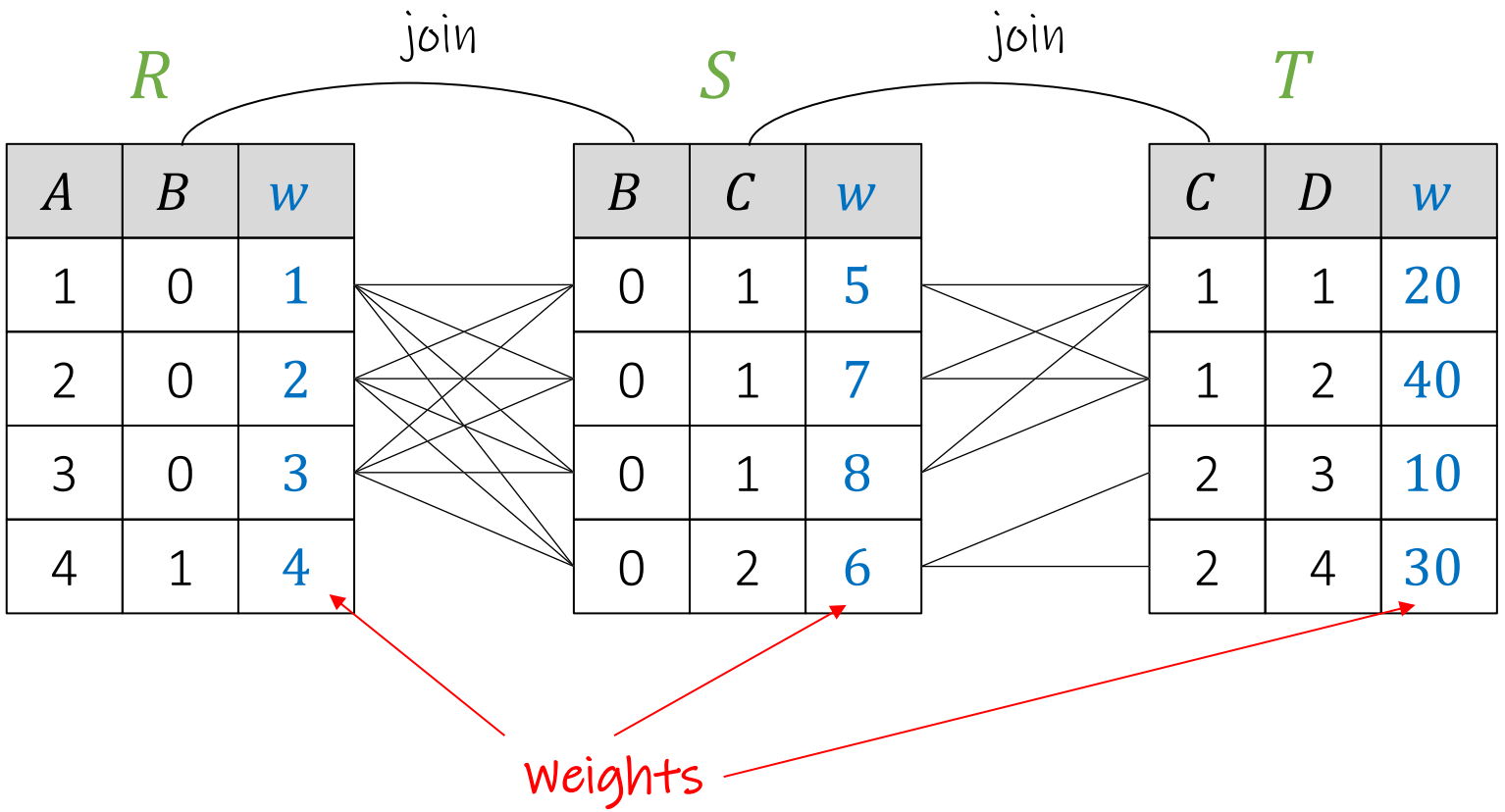

```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
limit    6
```

What do we get now?

Result

A	B	C	D	weight
1	0	2	3	17
2	0	2	3	18
3	0	2	3	19
1	0	1	1	26
2	0	1	1	27
3	0	1	1	28
1	0	1	1	28
...	...	...	...	...

Sorting & Top-*k* evaluation with SQL




```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
limit    6
```

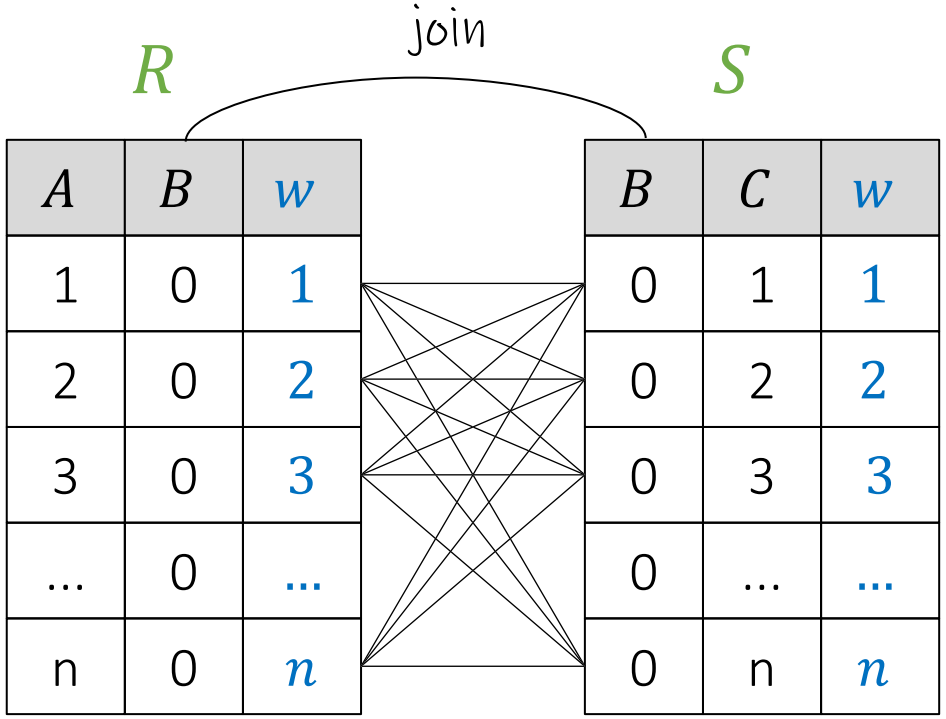
What do we get now?

Result

A	B	C	D	weight
1	0	2	3	17
2	0	2	3	18
3	0	2	3	19
1	0	1	1	26
2	0	1	1	27
3	0	1	1	28
1	0	1	1	28
...	...	...	...	...

Top-k is evaluated inefficiently by modern DBMS's

606



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

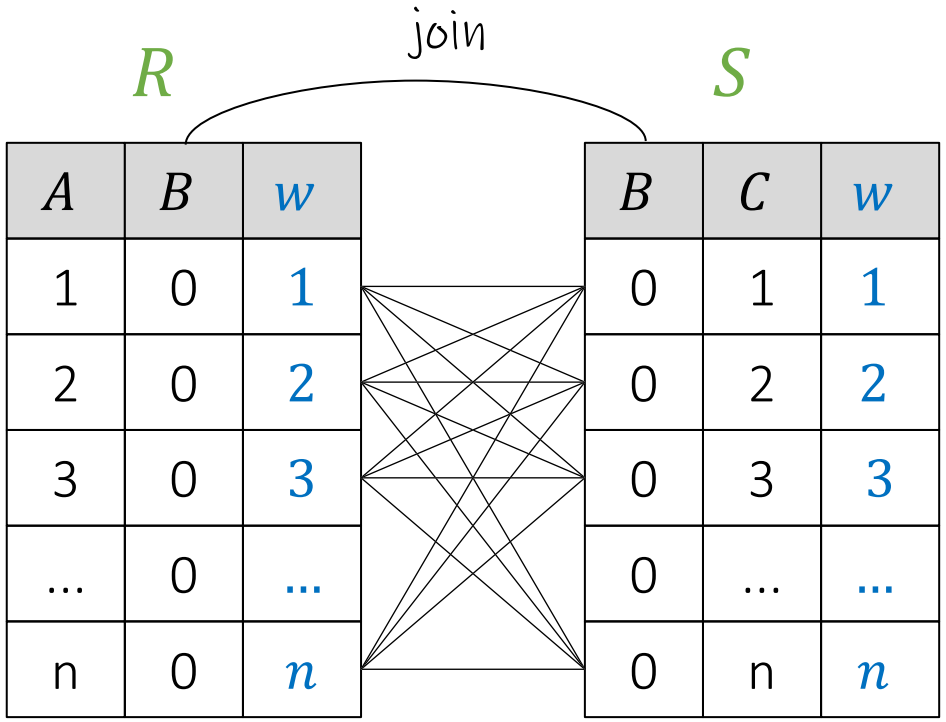
B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

```
select   A, R.B, S.C,
         R.w + S.w as weight
from     R, S
where    R.B=S.B
order by weight ASC
limit    1
```

What will this query return?

?

Top-k is evaluated inefficiently by modern DBMS's



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

```
select   A, R.B, S.C,
         R.w + S.w as weight
from     R, S
where    R.B=S.B
order by weight ASC
limit    1
```

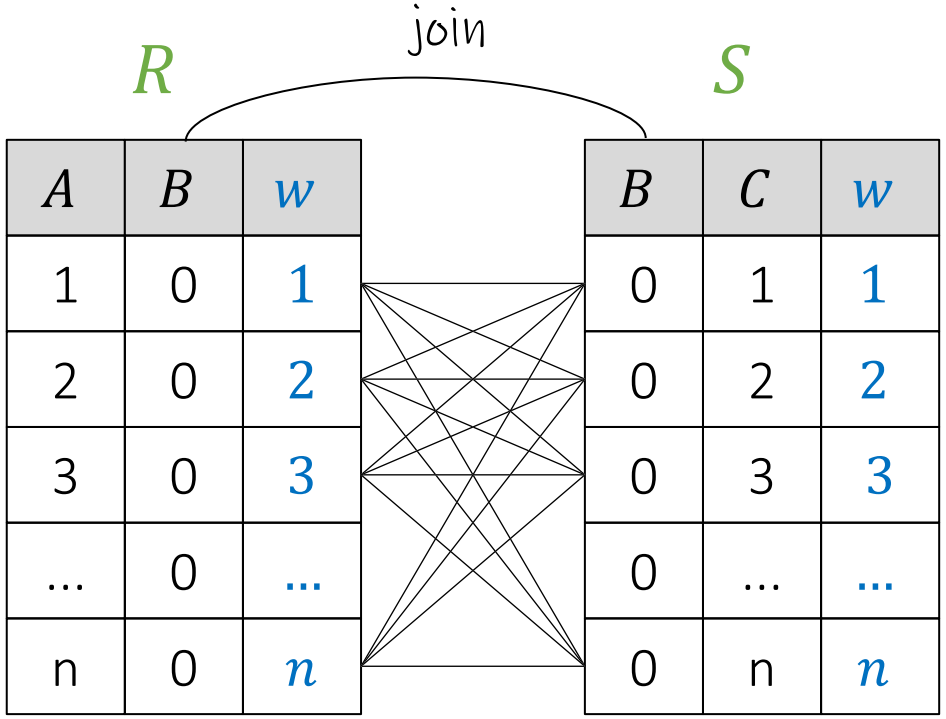
Result

A	B	C	weight
1	0	1	2
1	0	2	3
2	0	1	3
3	0	1	4
2	0	2	4
1	0	3	4
4	0	1	5
…	…	…	…
n	0	n	n*n

Can you see any possible problem of this query as n gets bigger?

?

Top-k is evaluated inefficiently by modern DBMS's



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

```
select   A, R.B, S.C,
         R.w + S.w as weight
from     R, S
where    R.B=S.B
order by weight ASC
limit    1
```

Result

A	B	C	weight
1	0	1	2
1	0	2	3
2	0	1	3
3	0	1	4
2	0	2	4
1	0	3	4
4	0	1	5
…	…	…	…
n	0	n	n*n

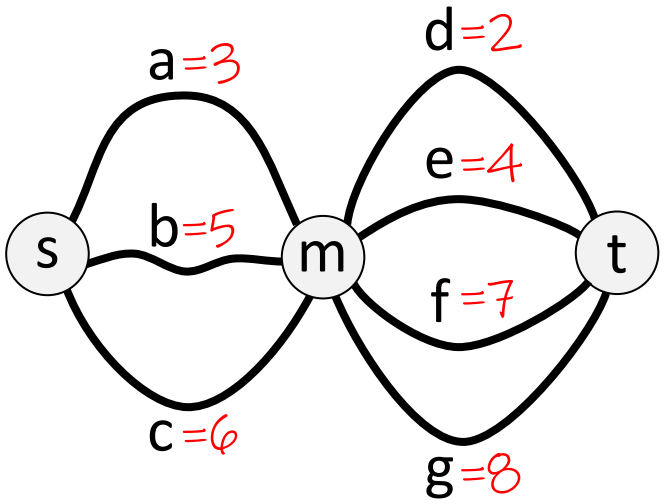
n^2 total results. But we are only interested in the top-1.

Problem: Database first calculates all n^2 results before sorting.

Question: is there any way to push the sorting behind the join?

?

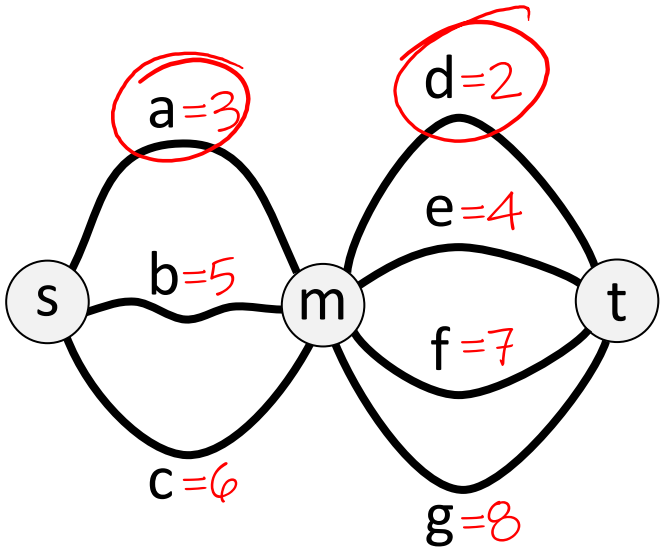
Digression: Distributivity = efficient factorization



What is the shortest path from s to t?

?

Digression: Distributivity = efficient factorization



What is the shortest path from s to t?

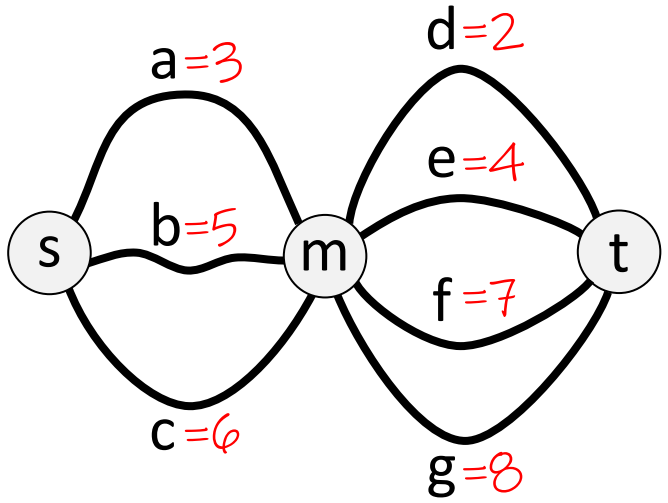
Answer: 5 = 3 + 2

min [a + d, a + e, a + f, a + g, ..., c + g]

min[3+2, 3+4, 3+7, 3+8, ..., 6+8]

?

Digression: Distributivity = efficient factorization



What is the shortest path from s to t?

Answer: 5 = 3 + 2

Principle of optimality from Dynamic Programming: *irrespective of the initial state and decision, an optimal solution continues optimally from the resulting state*

min [a + d, a + e, a + f, a + g, ..., c + g]

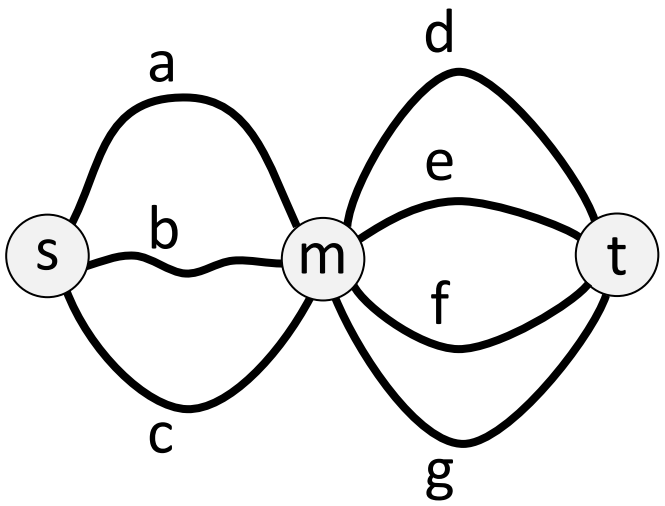
min[3+2, 3+4, 3+7, 3+8, ..., 6+8]

= min [a, b, c] + min [d, e, f, g]

min[3,5,6] + min[2,4,7,8]

min[x,y]+z = min[(x+z), (y+z)]
(+ distributes over min)

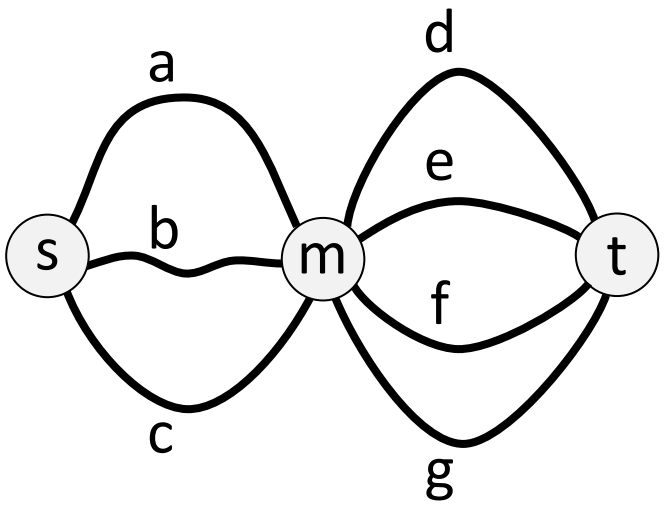
Digression: Distributivity = efficient factorization



How many paths are there from s to t?

?

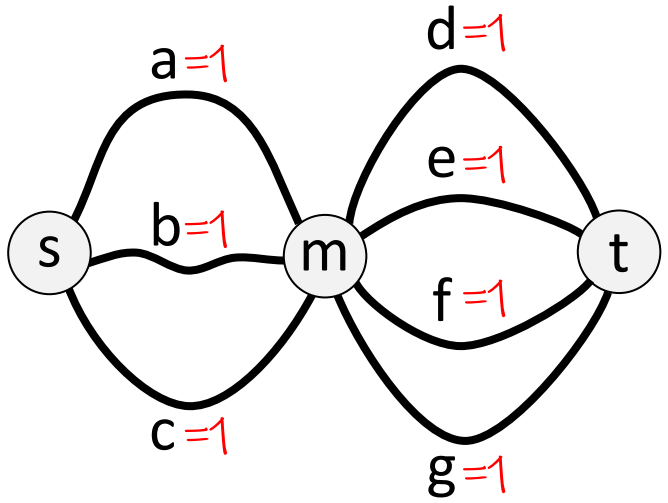
Digression: Distributivity = efficient factorization



How many paths are there from s to t?

Answer: $12 = 3 \cdot 4$

Digression: Distributivity = efficient factorization



How many paths are there from s to t?

Answer: 12 = 3 · 4

The more general algebraic structure behind these two examples are "semirings" (much more on those later in class)

count [a·d, a·e, a · f, a · g, …, c · g]

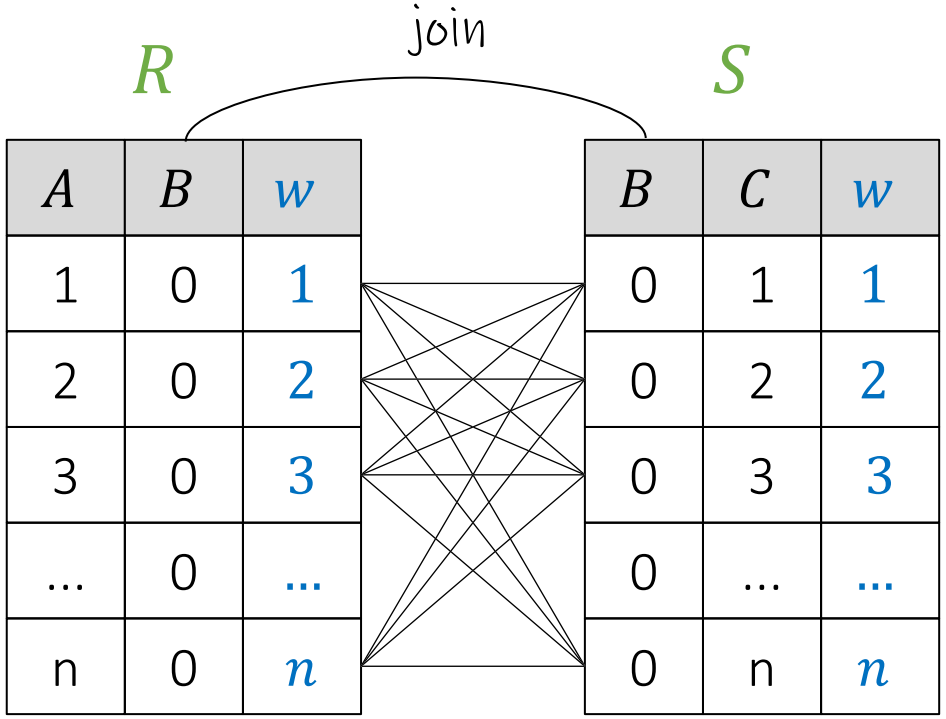
count[1·1, 1·1, 1·1, 1·1, …, 1·1] (12)

= count [a, b, c] · count [d, e, f, g]

count[1,1,1] · count[1,1,1,1]

+[x,y] · z = +[x·z,y·z]
(· distributes over +)

Top-k is evaluated inefficiently by modern DBMS's



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

```
-------------------------
-- Query 1
-------------------------
SELECT   A, R.B, S.C,
            R.W + S.W as weight
FROM      R, S
WHERE    R.B=S.B
ORDER BY weight ASC
LIMIT    1;
```

```
-------------------------
-- Query 2
-------------------------
```

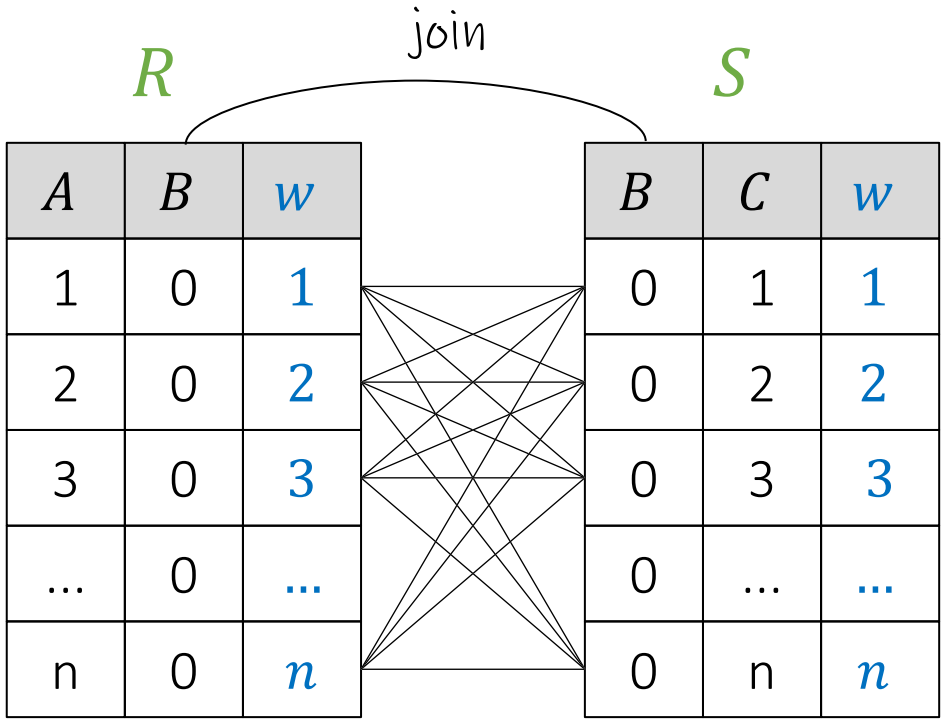
?

n=1000: t_{Q1}= 0.88 sec t_{Q2}=2 msec

n=5000: t_{Q1}=18.6 sec t_{Q2}=8 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Top-k is evaluated inefficiently by modern DBMS's



Maximal intermediate result size is O(n) ☺ What is this algorithm called?

?

```
-------------------------
-- Query 1
-------------------------
SELECT   A, R.B, S.C,
           R.W + S.W as weight
FROM      R, S
WHERE    R.B=S.B
ORDER BY weight ASC
LIMIT    1;
```

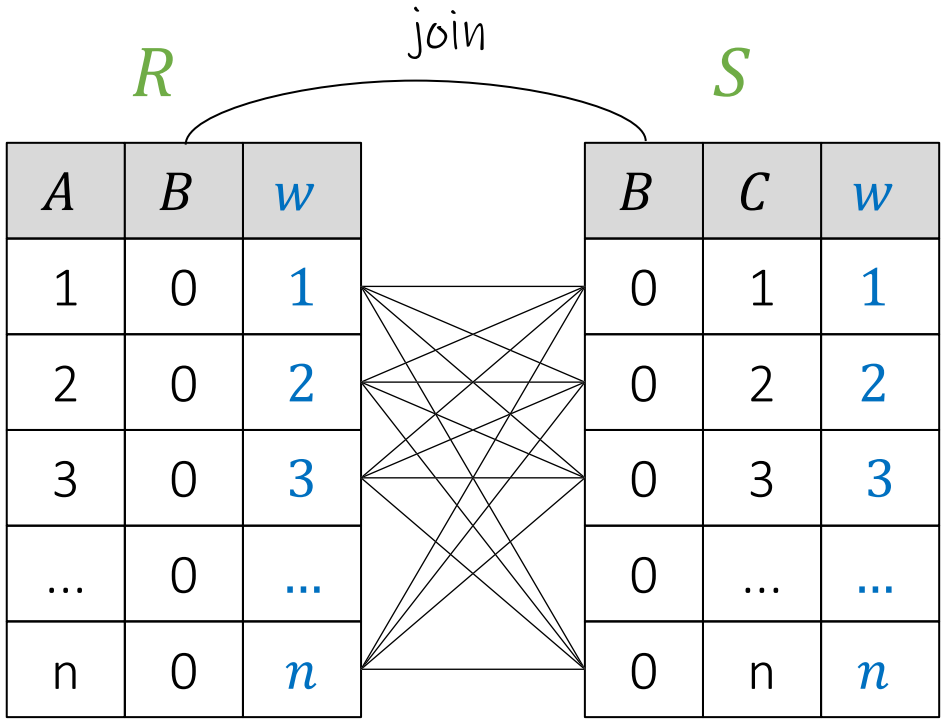
```
-------------------------
-- Query 2
-------------------------
SELECT R.A, X.B, S.C, X.W as weight
FROM R, S,
     (SELECT T1.B, W1, W2, W1+W2 W
     FROM
          (SELECT B, MIN(W) W1
          FROM R
          GROUP BY B) T1,
          (SELECT B, MIN(W) W2
          FROM S
          GROUP BY B) T2
     WHERE T1.B = T2.B
     ORDER BY W ASC
     LIMIT 1) X
WHERE X.B = R.B
AND X.W1 = R.W
AND X.B = S.B
AND X.W2 = S.W
LIMIT   1;
```

n=1000: t_{Q1}= 0.88 sec t_{Q2}=2 msec

n=5000: t_{Q1}=18.6 sec t_{Q2}=8 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Top-k is evaluated inefficiently by modern DBMS's



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

Maximal intermediate result size is O(n) ☺
What is this algorithm called?

Dynamic programming

```
-------------------------
-- Query 1
-------------------------
SELECT   A, R.B, S.C,
           R.W + S.W as weight
FROM      R, S
WHERE    R.B=S.B
ORDER BY weight ASC
LIMIT    1;
```

$O(n^2)$

n=1000: t_{Q1}= 0.88 sec

n=5000: t_{Q1}=18.6 sec

```
-------------------------
-- Query 2
-------------------------
SELECT R.A, X.B, S.C, X.W as weight
FROM R, S,
     (SELECT T1.B, W1, W2, W1+W2 W
     FROM
          (SELECT B, MIN(W) W1
          FROM R
          GROUP BY B) T1,
          (SELECT B, MIN(W) W2
          FROM S
          GROUP BY B) T2
     WHERE T1.B = T2.B
     ORDER BY W ASC
     LIMIT 1) X
WHERE X.B = R.B
AND X.W1 = R.W
AND X.B = S.B
AND X.W2 = S.W
LIMIT   1;
```

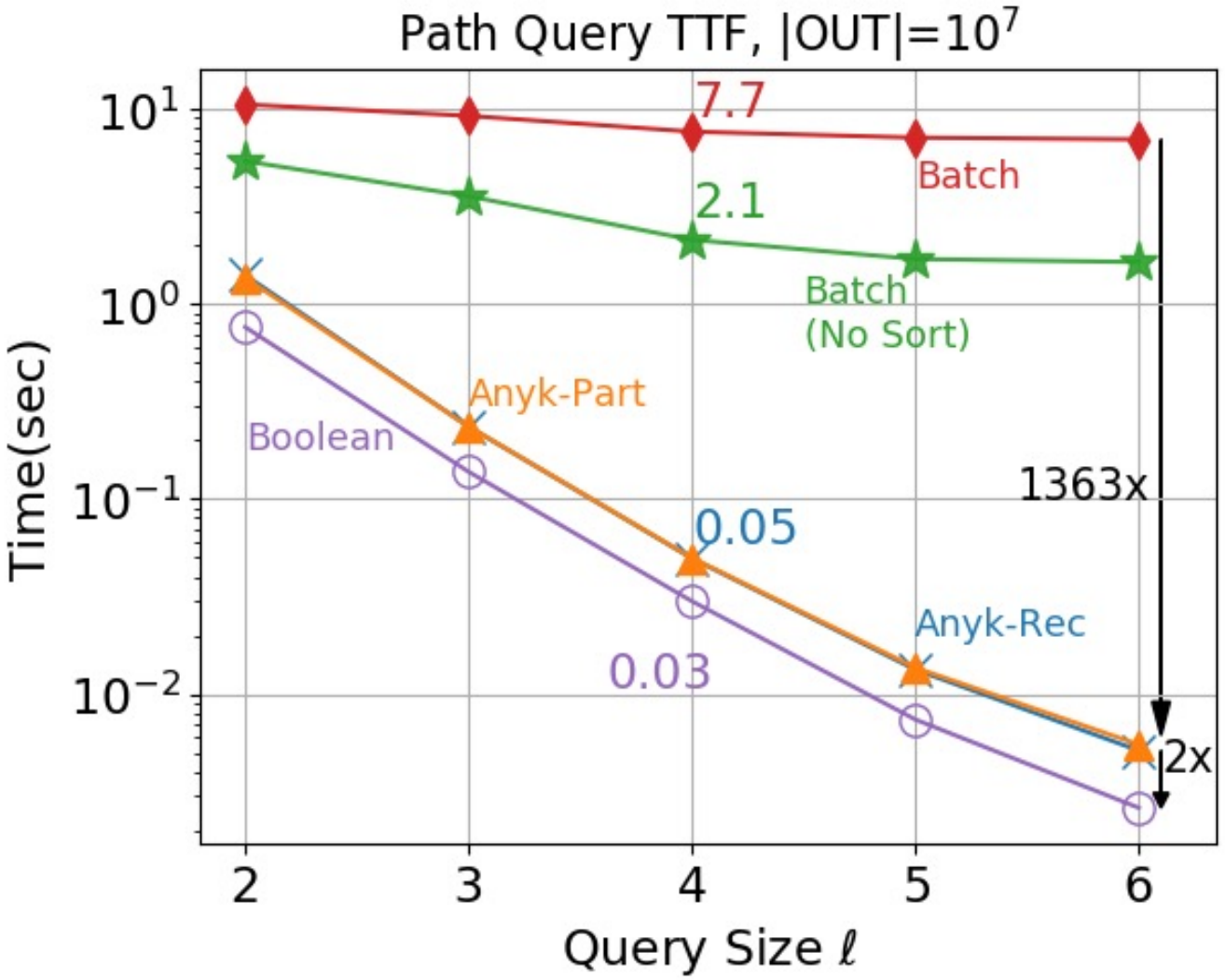
$O(n)$

t_{Q2}=2 msec

t_{Q2}=8 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Any-*k*: Faster and more versatile than Top-*k*



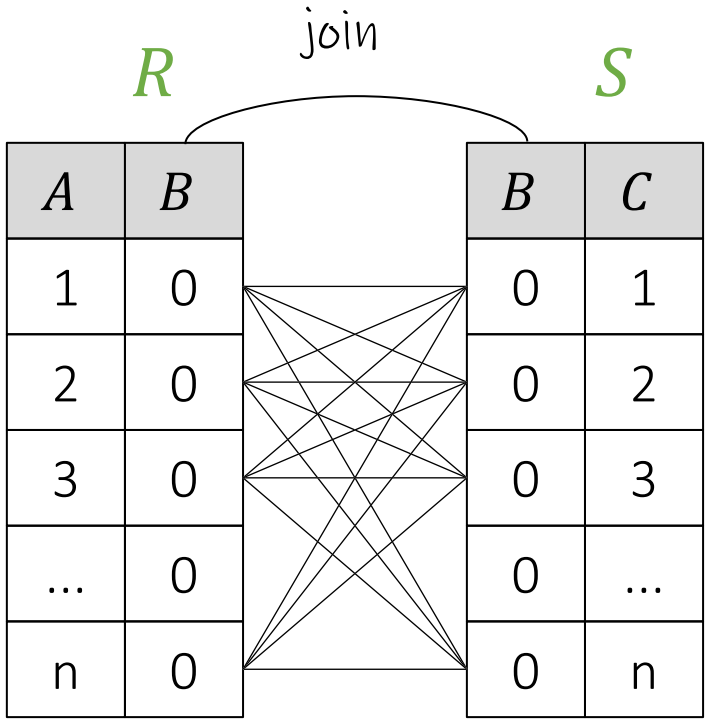
Path query with
constant size output
and increasing query size

Tziavalis+ [PVLDB'20], Tziavelis+ [SIGMOD'20 tutorial]

https://northeastern-datalab.github.io/anyk/
https://northeastern-datalab.github.io/topk-join-tutorial/
https://www.youtube.com/watch?v=KpUQayBuaQI&list=PL_72ERGKF6DR7kvGNwwjWlbpScKtGjt9R&index=2

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB 2020. http://www.vldb.org/pvldb/vol13/p1582-tziavelis.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Even Grouping Aggregates can be improved




```
-------------------------
-- Query 1
-------------------------
SELECT   count(*) C
INTO record1
FROM      R, S
WHERE    R.B=S.B;
```

```
-------------------------
-- Query 2
-------------------------
SELECT SUM(C) as C
INTO record2
FROM
    (SELECT T1.B, C1, C2, C1*C2 C
    FROM
        (SELECT B, COUNT(*) C1
        FROM R
        GROUP BY B) T1,
        (SELECT B, COUNT(*) C2
        FROM S
        GROUP BY B) T2
    WHERE T1.B = T2.B) X;
```

n=1000: t_{Q1}= 0.374 sec t_{Q2}=3 msec

n=5000: t_{Q1}= 10.021 sec t_{Q2}=5 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql