Computation of Aggregations

- Part of the data warehouse loading involves computation of aggregations → materialized views
  - Order in which they are computed can be important
  - Sort-based or Hash-based

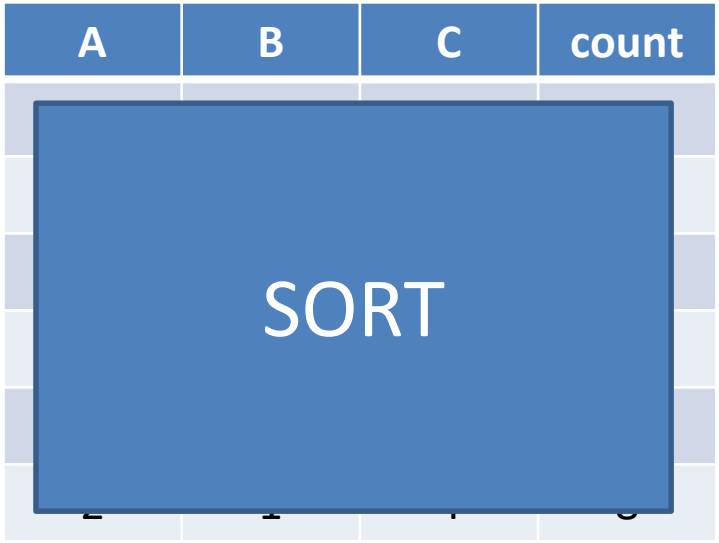
See, e.g.: Agarwal et al. On the computation of multidimensional aggregates. VLDB 1996

Sort-Based Aggregation

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

```
SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;
```

Sort-Based Aggregation



SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	8

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	8

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	17

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	17
3	3	3	5

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	17
3	3	3	5

Sort-Based Aggregation

- Observation:
  - Table sorted on ABC

    = sorted on AB

    = sorted on A
  - One sort supports 3 aggregations

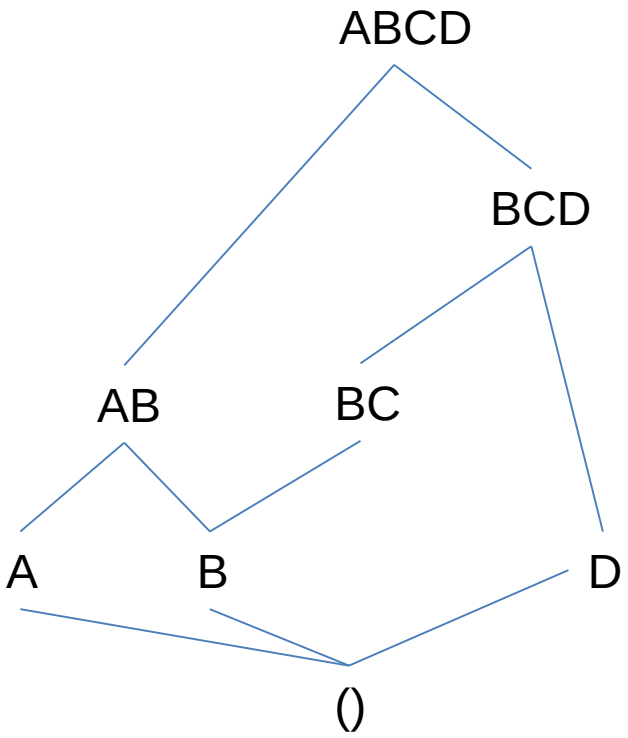
A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

Pipe-Sort

- Sort on ABC
- Scan sorted relation
  - As long as next tuple is the same as previous
    - Update aggregated tuple; add count of current
  - Else
    - Ship aggregated tuple to disk
    - Pipe tuple to procedure computing aggregation on AB

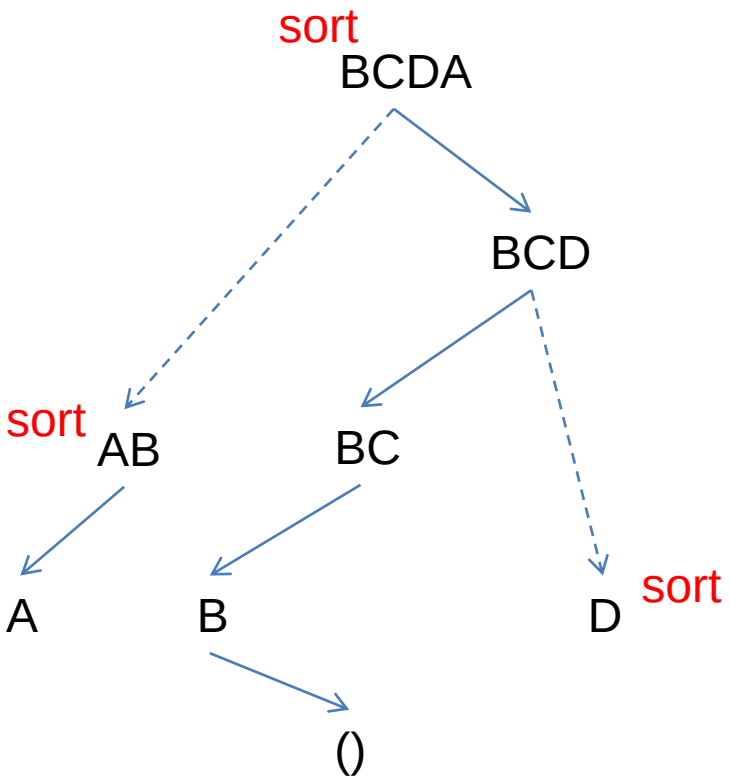
Pipe-Sort

- Key problem: divide materialized views lattice into “pipes”, minimizing sorts



Pipe-Sort

- Key problem: divide materialized views lattice into “pipes”, minimizing sorts



Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	8

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	8
2	1	4	9

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	8
2	1	4	9
1	8	6	10

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	**14**
2	1	4	9
1	8	6	10

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	14
2	1	4	9
1	8	6	10
3	3	3	5

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	14
2	1	4	**17**
1	8	6	10
3	3	3	5

Hash-Based Optimization

- Multiple hash tables may fit at the same time

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

A	B	C	SUM
1	5	6	14
2	1	4	17
1	8	6	10
3	3	3	5

A	C	SUM
1	6	24
2	4	17
3	3	5

C	SUM
6	24
4	17
3	5