COMP207
Database Development

Lecture 13

Query Processing:
Query Plans and their Execution

Semijoin ($\ltimes$)

- $R_1 \ltimes R_2$ = tuples from R_1 matching tuples in R_2

Two tuples match
if they have the same values
for all common attributes

Modules **Lecturers**

```
SELECT *
FROM Modules
WHERE EXISTS    (SELECT 1
                 FROM Lecturers
                 WHERE
                 Modules.module =
                 Lecturers.module)
```

=

module	year
COMP105	1

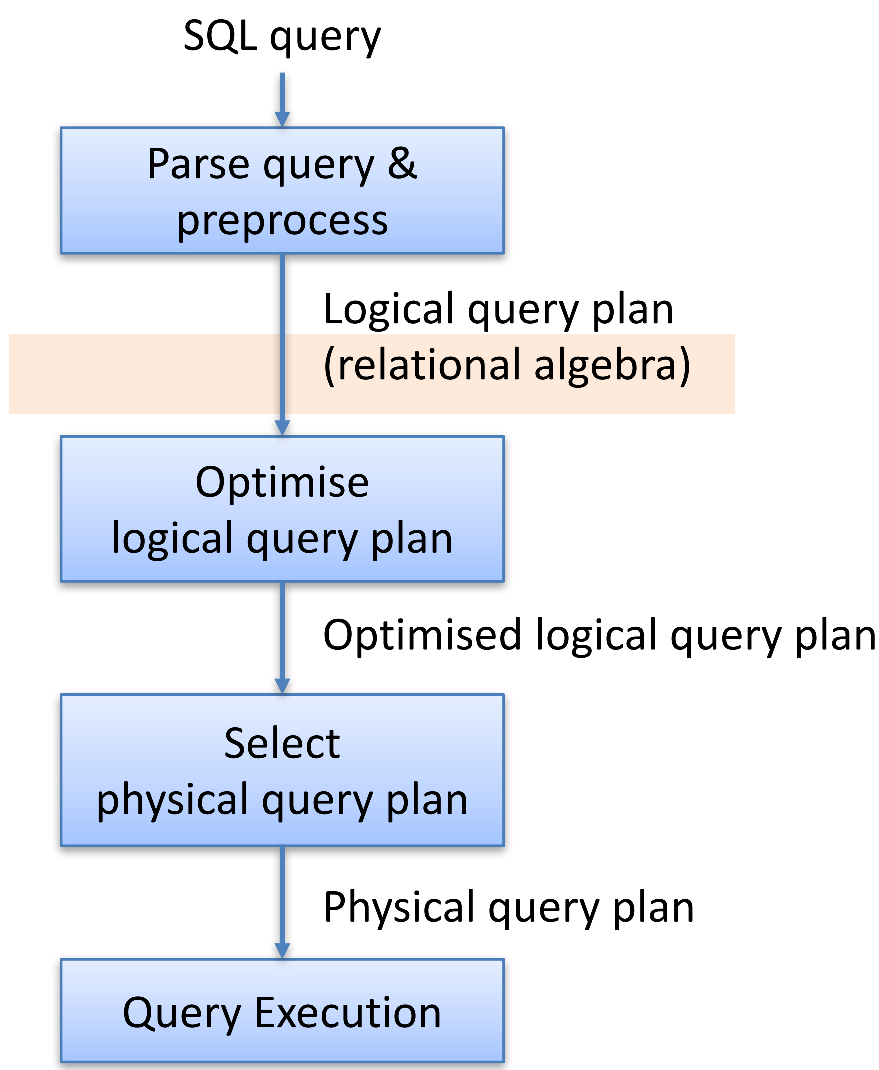
- Can be expressed by the other operators:

Modules $\ltimes$ **Lecturers** =

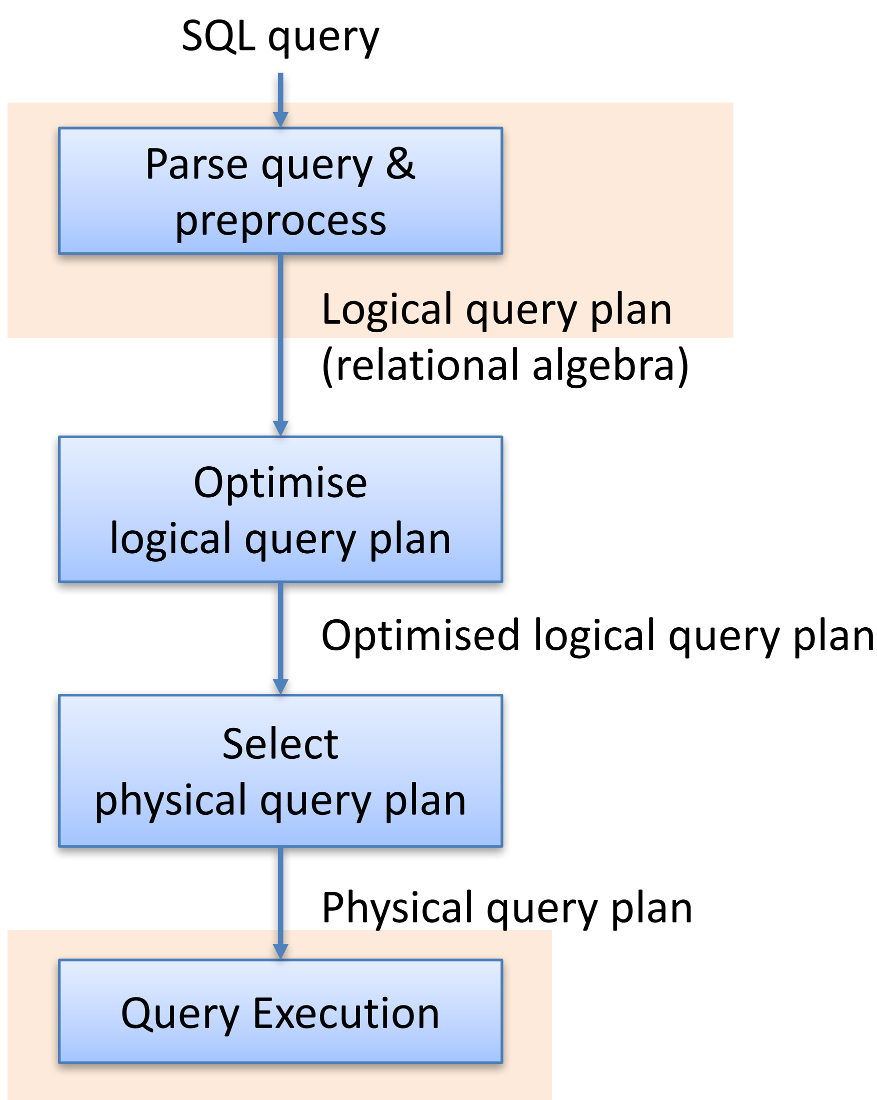
$$\pi_{\text{module,year}}(\sigma_{\text{module=module'}}(\text{Modules} \times \rho_{\text{module}\rightarrow\text{module'}}(\text{Lecturers})))$$

Reminder: Query Processing

Previous lecture



Reminder: Query Processing



This lecture

Query Plans

(a.k.a. Query Trees)

Query Plans

- A **relational algebra expression** that is obtained from an SQL query is also called a **(logical) query plan**

```
SELECT module, name
FROM Modules, Lecturers
WHERE code=module AND year=2;
```

SQL query

$$\pi_{\text{module,name}}(\sigma_{\text{code=module AND year=2}}(\text{Modules} \times \text{Lecturers}))$$

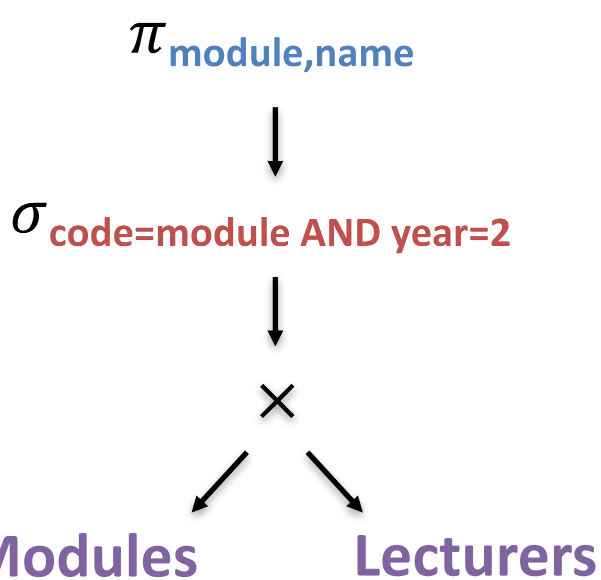
Query plan for the query

- Query plans are typically **represented as trees**

Query Plans As Trees

$$\pi_{\text{module,name}}(\sigma_{\text{code=module AND year=2}}(\text{Modules} \times \text{Lecturers}))$$

- Tree representation:



- Inner nodes = operators
- Leaves = input relations

- Such trees are evaluated from the leaves to the root

Exercise

- Represent the following query plan as a tree:

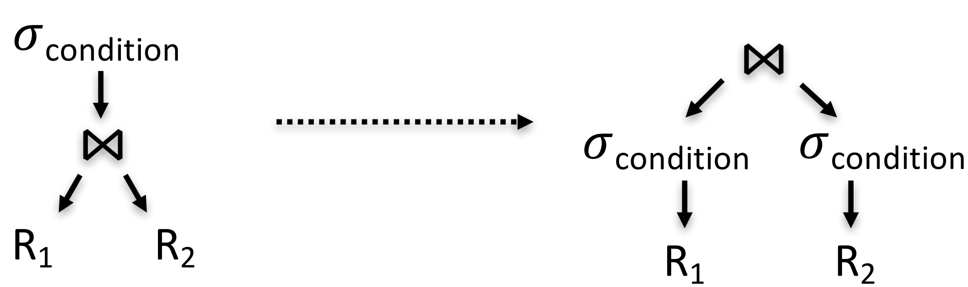
$$\sigma_{code=module}(\pi_{code}(\textbf{Modules}) \times \pi_{module,name}(\textbf{Lecturers}))$$

$$\begin{tikzcd}
& \sigma_{code=module} \arrow[d] & \\
& \times \arrow[dl] \arrow[dr] & \\
\pi_{code} \arrow[d] & & \pi_{module,name} \arrow[d] \\
\textbf{Modules} & & \textbf{Lecturers}
\end{tikzcd}$$

Equivalent Query Plans

- There are typically **many different query plans**
- DBMSs aim to **select a best possible query plan**
- Relational algebra is better suited than SQL for this
  - Can use **equivalence laws** of relational algebra to generate a query plan for the same query that can be executed faster!
  - Example:
    - $\sigma_{\text{condition}}(R_1 \bowtie R_2) = \sigma_{\text{condition}}(R_1) \bowtie \sigma_{\text{condition}}(R_2)$

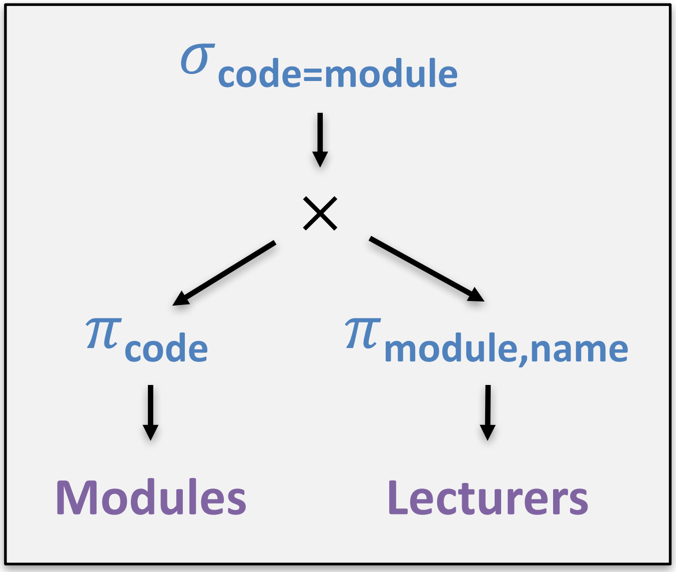
Details will come later…



Execution of Query Plans

Executing Query Plans

- Query plans tells us exactly how to compute the result to a query
- Proceed from bottom to top:
  - Compute an **intermediate result** for each node
  - For a **leaf** labeled with relation **R**, the intermediate result is **R**.
  - For an **inner node** labeled with operator **op**, get the intermediate result by applying **op** to the childrens’ intermediate results.
  - **Result of the query** = intermediate result of the root



Example

Final result

module	year	name
COMP201	2	S. Coope

$\sigma_{\textbf{year=2}}$

$\bowtie$

module	year	name
COMP105	1	J. Fearnley
COMP201	2	S. Coope

module	year
COMP105	1
COMP201	2

$\rho_{\textbf{code}\rightarrow\textbf{module}}$

$\pi_{\textbf{name, module}}$

name	module
J. Fearnley	COMP105
S. Coope	COMP201

Modules

code	year
COMP105	1
COMP201	2

Lecturers

name	module	phone
J. Fearnley	COMP105	54265
S. Coope	COMP201	54232

Exercise (5 min)

Execute this query plan:

Final result

A	B	C	D
1	2	2	3
2	5	5	7

$\sigma_{B=C}$

A	B	C	D
1	2	3	4
1	2	2	3
2	5	5	7

$\bowtie$

A	B
1	2
2	5

$\pi_{A,B}$

$\rho_{B \to C, C \to D}$

A	C	D
1	3	4
1	2	3
2	5	7

R

A	B	C	D
1	2	4	3
1	2	1	4
2	5	7	1

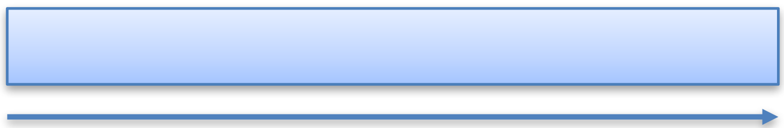
S

A	B	C
1	3	4
1	2	3
2	5	7

How to ”Apply” An Operator?

- How to compute $\sigma_{condition}(R)$?

R:



have to read entire file

Is there a faster way?

```
for each tuple t in R:
    if t satisfies condition:
        output t
```

- How to compute $\pi_{attribute\ list}(R)$?

  Similar! Read R only once…

```
for each tuple t in R:
    output the restriction
        of t to the attributes
        in attribute list
```

Is there a faster way?

- How to compute $R \bowtie S$?

  Many different ways…

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
    for each tuple s in S:
        if r and s have the same values for all common attributes:
            output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley

r ⋈ s

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
    for each tuple s in S:
        if r and s have the same values for all common attributes:
            output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
  for each tuple s in S:
    if r and s have the same values for all common attributes:
      output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
    for each tuple s in S:
        if r and s have the same values for all common attributes:
            output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
  for each tuple s in S:
    if r and s have the same values for all common attributes:
      output r ⋈ s the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
    for each tuple s in S:
        if r and s have the same values for all common attributes:
            output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley
COMP201	2	S. Coope

r ⋈ s

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
  for each tuple s in S:
    if r and s have the same values for all common attributes:
      output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley
COMP201	2	S. Coope

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:

    for each tuple s in S:

        if r and s have the same values for all common attributes:

            output r ⋈ s

the tuple obtained by joining **r** and **s**

Modules

module	year
COMP105	1
COMP201	2

r

⋈

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

s

=

module	year	name
COMP105	1	J. Fearnley
COMP201	2	S. Coope
COMP201	2	J. Smith

r ⋈ s

Naïve Computation of Joins

- Nested Loop Join Algorithm:

Compute R ⋈ S:

for each tuple r in R:
  for each tuple s in S:
    if r and s have the same values for all common attributes:
      output r ⋈ s

- Slow: for each tuple r in R reads entire relation S
- Running time: $O(|\mathbf{R}| \times |\mathbf{S}|)$

Number of tuples in **R**

Number of tuples in **S**

Can We Go Faster?

Yes, we can!

Equijoins

- Equijoin $R \bowtie_{A=B} S$ is defined as $\sigma_{A=B}(R \times S)$

A, B are the **join attributes**

Modules

code	year
COMP105	1
COMP201	2

Lecturers

name	module
J. Fearnley	COMP105
S. Coope	COMP201
J. Smith	COMP201

Modules $\bowtie_{code=module}$ Lecturers

code	year	name	module
COMP105	1	J. Fearnley	COMP105
COMP201	2	S. Coope	COMP201
COMP201	2	J. Smith	COMP201

- *If* R is sorted on A *and* S is sorted on B, *then* $R \bowtie_{A=B} S$ can be computed with one pass over R and S

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…

sorted

sorted

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…

sorted

sorted

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…
4	4	52	74	…

sorted

sorted

Merging

as in merge sort

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…
4	4	52	74	…

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…
4	4	52	74	…
4	4	52	9	…

sorted

sorted

Merging

as in merge sort

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

Running time: $O(|\mathbf{R}| + |\mathbf{S}|)$

R

A	C	…
1	5	…
1	4	…
2	99	…
3	15	…
4	52	…
…	…	…

sorted

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

sorted

=

A	B	C	D	…
2	2	99	50	…
2	2	99	94	…
2	2	99	11	…
4	4	52	74	…
4	4	52	9	…
…	…	…	…	…

- What if a value occurs twice or more in column A?

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…

Remember tuples in S that match with the current value of A

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
$\boxed{2}$	$\boxed{4}$	$\boxed{…}$
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
$\boxed{2}$	$\boxed{50}$	$\boxed{…}$
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute R $\bowtie_{A=B}$ S

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…

Remember tuples in S that match with the current value of A

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…
2	2	4	11	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…
2	2	4	11	…

Remember tuples in S that match with the current value of A

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…
2	2	4	11	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute $R \bowtie_{A=B} S$

Assume: R is sorted on A and S is sorted on B

R

A	C	...
1	5	...
2	4	...
2	99	...
2	15	...
4	52	...
...	...	...

duplicates

$\bowtie_{A=B}$

S

B	D	...
2	50	...
2	94	...
2	11	...
4	74	...
4	9	...
...	...	...

=

A	B	C	D	...
2	2	4	50	...
2	2	4	94	...
2	2	4	11	...
2	2	99	50	...

Remember tuples in S that match with the current value of A

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…
2	2	4	11	…
2	2	99	50	…

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

R

A	C	...
1	5	...
2	4	...
2	99	...
2	15	...
4	52	...
...	...	...

duplicates

$\bowtie_{A=B}$

S

B	D	...
2	50	...
2	94	...
2	11	...
4	74	...
4	9	...
...	...	...

=

A	B	C	D	...
2	2	4	50	...
2	2	4	94	...
2	2	4	11	...
2	2	99	50	...
2	2	99	94	...

Remember tuples in S that match with the current value of **A**

Merging With Duplicates in Column **A**

- Goal: compute **R** $\bowtie_{A=B}$ **S**

Assume: **R** is sorted on **A** and **S** is sorted on **B**

What is the running time?

R

A	C	…
1	5	…
2	4	…
2	99	…
2	15	…
4	52	…
…	…	…

duplicates

$\bowtie_{A=B}$

S

B	D	…
2	50	…
2	94	…
2	11	…
4	74	…
4	9	…
…	…	…

=

A	B	C	D	…
2	2	4	50	…
2	2	4	94	…
2	2	4	11	…
2	2	99	50	…
2	2	99	94	…
…	…	…	…	…

Remember tuples in S that match with the current value of **A**

Faster Joins With Sorting

- Sort Join Algorithm:

Compute R $\bowtie_{A=B}$ S:

1. Sort R on A
2. Sort S on B
3. Merge the sorted R and S

Running time: $O(|R| \times \log_2 |R|)$

Running time: $O(|S| \times \log_2 |S|)$

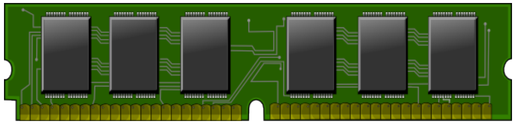
Running time: $O(|R| + |S|)$

- Typical running time: $O(|R|\log_2|R| + |S|\log_2|S|)$
  - If not “too many” values in A occur multiple times
  - E.g., this is the case if A is a key
- Typically much faster than Nested Loop Join

Remarks

- Various **join algorithms** in practice:
  - Index joins
  - Hash joins
  - Multiway joins: join more than two relations at once
- Can compute **other operations of relational algebra** using similar methods as those in this lecture
- **We've neglected that relations are stored on disk**

Running Time vs Disk Accesses

fast   slow

Relevant parameters:

- **B** = size of a disk block (typically 512→4096 bytes)
- **M** = number of disk blocks that fit into available RAM

Algorithm	No. of elementary operations	No. of disk accesses
Reading a relation **R**	$O(\lvert R \rvert)$	$O\left(\frac{\lvert R \rvert}{B}\right)$
Sorting **R** on attribute **A**	$O(\lvert R \rvert \log_2 \lvert R \rvert)$	$O\left(\frac{\lvert R \rvert}{B} \log_M \frac{\lvert R \rvert}{B}\right)$

External memory merge sort

Summary

- Query plans are evaluated bottom-up – from the leaves to the root
- Each operator can be computed in different ways
  - Selection: e.g., linear scans (reading the relation once)
  - Projection: linear scans
  - Joins: Nested Loop Join, Sort Join, Index Join, Hash Join, …
- Next lecture: faster query processing with indexes

Know How

Can you help us?

Please complete this survey to help with our research. Your answers will help us improve our service for you.

Use this short URL or the QR code

https://tinyurl.com/y5cot2aj

Open the camera on your phone to use the QR code