Introduction to Database Systems
CSE 414

Lecture 28: Intro to Query Optimization

Final Exam

- Thursday 6/7, 2:30-4:20pm
- Location: here
- Comprehensive exam
  - Covers all lectures, sections, web quizzes, HWs, and readings
- Can bring 2 letter-size sheets of notes
  - Handwritten or printed
- More info on course website
- Review session:
  - Sunday 6/3, 2:30-5pm, SMI 102

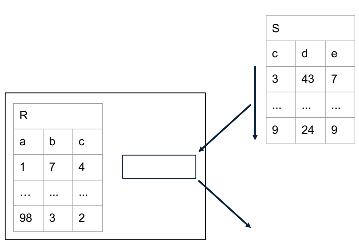
Big Picture

- How to choose the "best" query plan to run? (aka query optimization)
- To answer this question we need to understand:
  - Data organization on the disk
  - Index structures and how they are used in queries
  - A way to model query "costs"
  - Compute cost for each query operator
  - Compute cost for each physical plan

Last topics this quarter!

Review: Join Algorithms

- Nested loop join
- Hash join
- Sort-merge join



Hash Join

Hash Join

Hash join: R ⋈ S

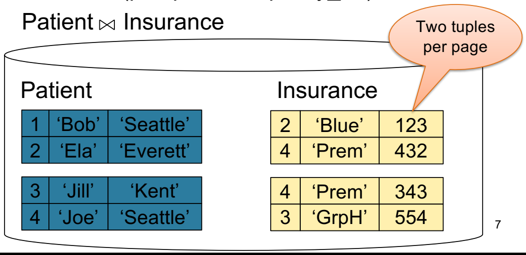
- Scan R, build hash table in main memory
- Then scan S and join
- Cost: B(R) + B(S)
- Which relation to build the hash table on?
- One-pass algorithm when B(R) ≤ M
  - M = number of memory pages available

Hash Join Example

Patient(pid, name, address)

Insurance(pid, provider, policy_nb)

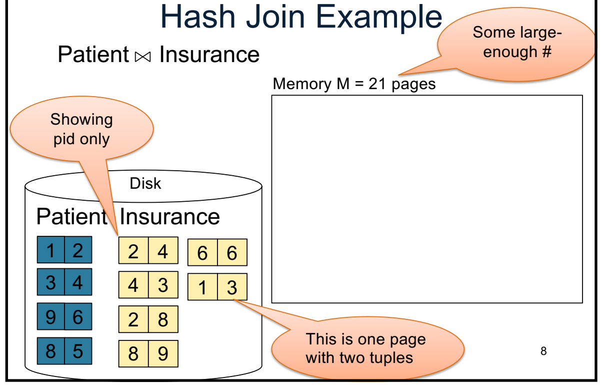
Patient ⋈ Insurance



7

Hash Join Example

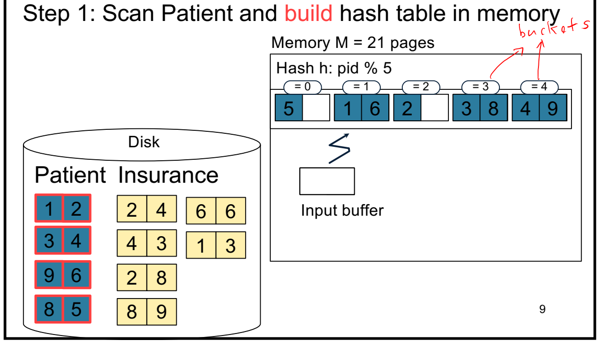
Patient ⋈ Insurance



8

Hash Join Example

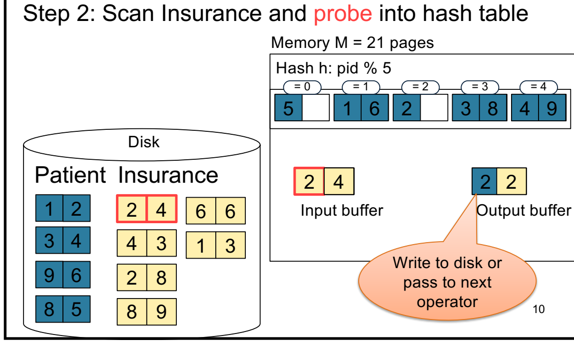
Step 1: Scan Patient and build hash table in memory



9

Hash Join Example

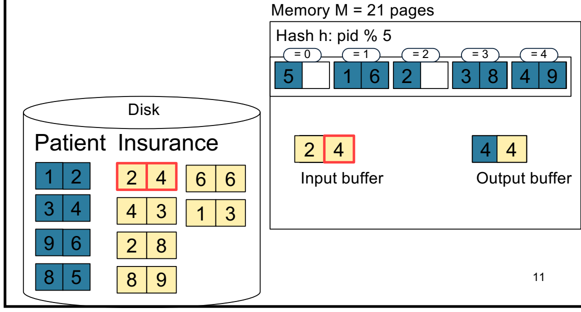
Step 2: Scan Insurance and probe into hash table



10

Hash Join Example

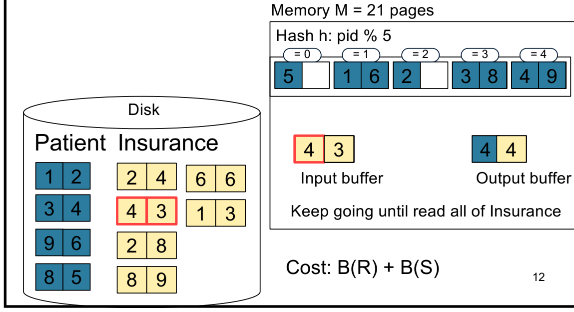
Step 2: Scan Insurance and probe into hash table



11

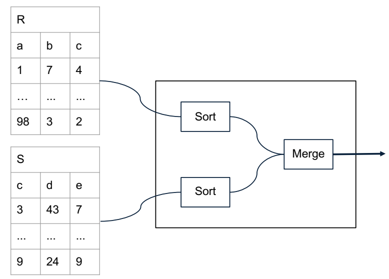
Hash Join Example

Step 2: Scan Insurance and probe into hash table



Cost: B(R) + B(S)

12



Sort-Merge Join

Sort-Merge Join

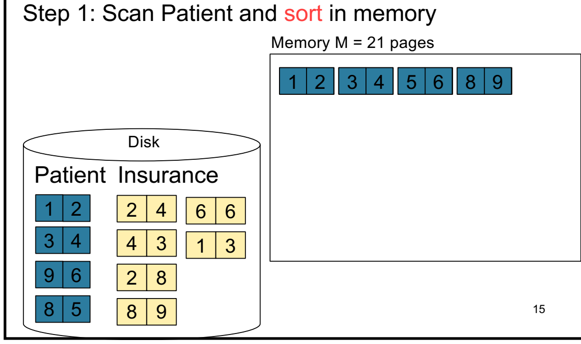
Sort-merge join: R ⋈ S

- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

- Cost: B(R) + B(S)
- One pass algorithm when B(S) + B(R) <= M
- Typically, this is NOT a one pass algorithm

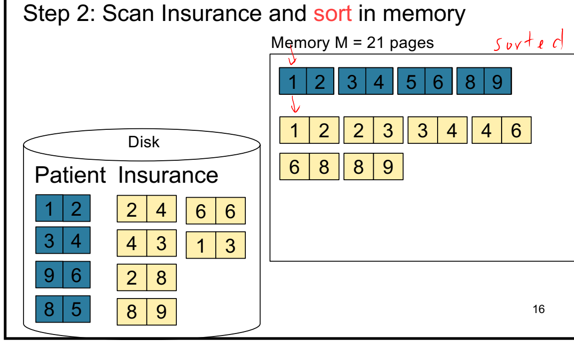
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory



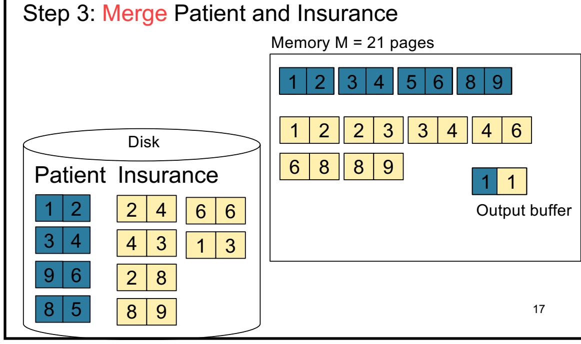
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory



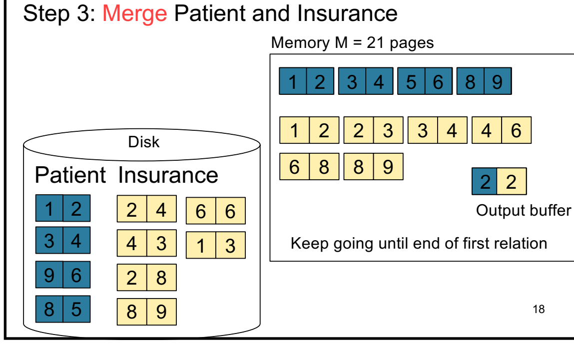
Sort-Merge Join Example

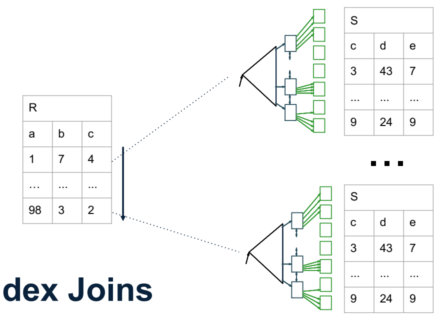
Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 3: Merge Patient and Insurance





Index Joins

Index Nested Loop Join

R ⋈ S

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

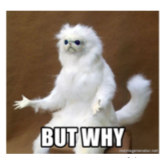
```
for r in R
  // use index to lookup
  for s' in S that should be joined with r
    s = fetch S tuple pointed to by s' from disk
    output (r,s)
```


Index Nested Loop Join

R ⋈ S

```
for r in R
  // use index to lookup
  for s' in S that should be joined with r
    s = fetch S tuple pointed to by s' from disk
    output (r,s)
```

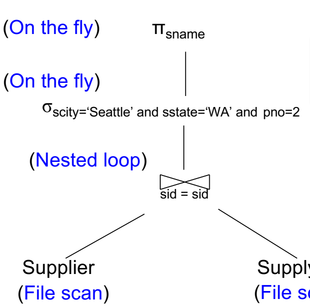
- Cost:
  - If index on S is clustered:
    B(R) + T(R) * (B(S) * 1/V(S,a))
  - If index on S is unclustered:
    B(R) + T(R) * (T(S) * 1/V(S,a))



Review:
Logical vs Physical Plans

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Review: Physical Query Plan 1

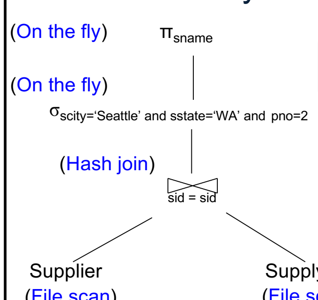


A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```


Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Review: Physical Query Plan 2

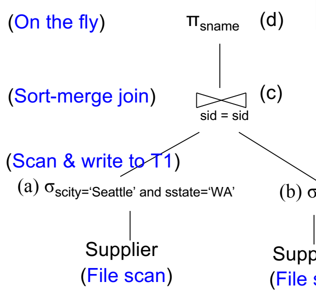


Same logical query plan
Different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Review: Physical Query Plan 3



Different but equivalent logical query plan; different physical

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

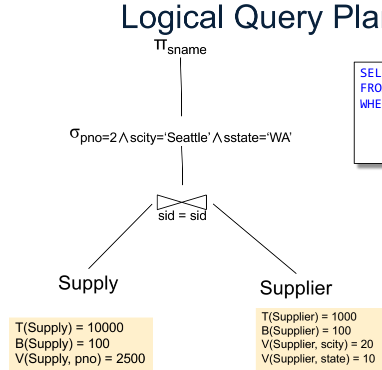

Query Optimization: Overview

- Compute cost of each operator, which depends on:
  - Table statistics (# of tuples produced)
  - Algorithm used to implement each operator
- Cost of a physical plan = sum(each operator cost)
- Cost each plan and choose the one with lowest cost

Estimating Table Statistics

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

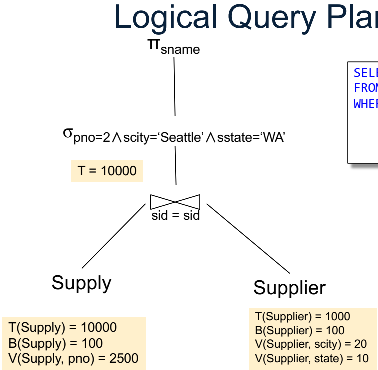



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

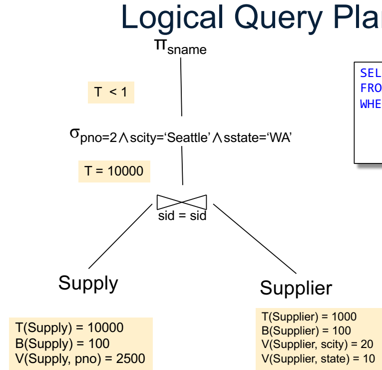



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

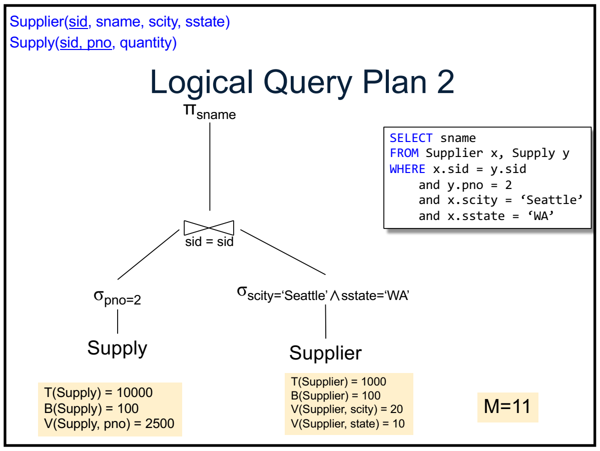
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

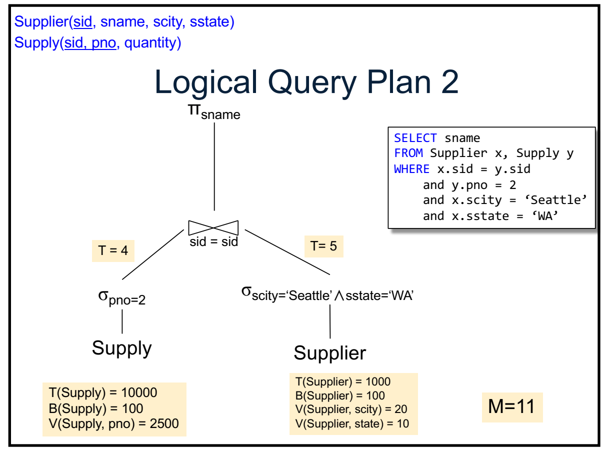
Logical Query Plan 1

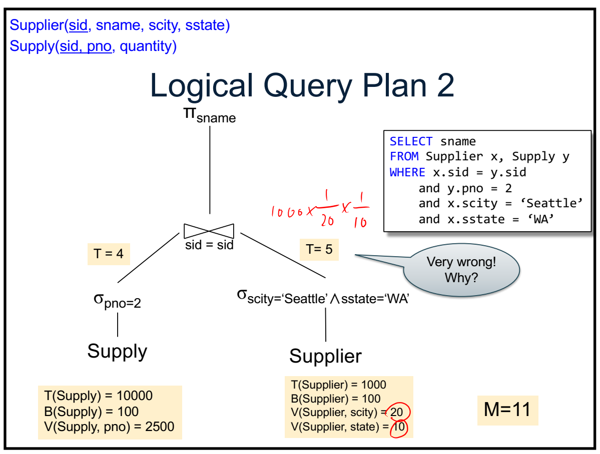


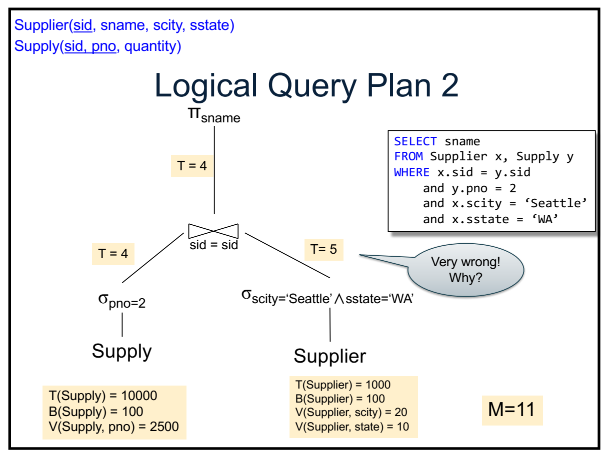

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

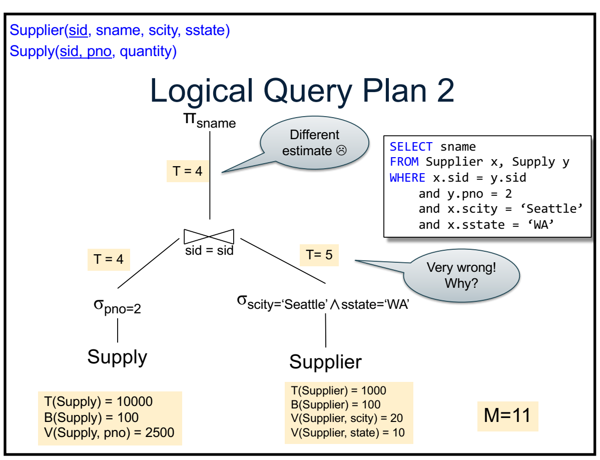
M=11



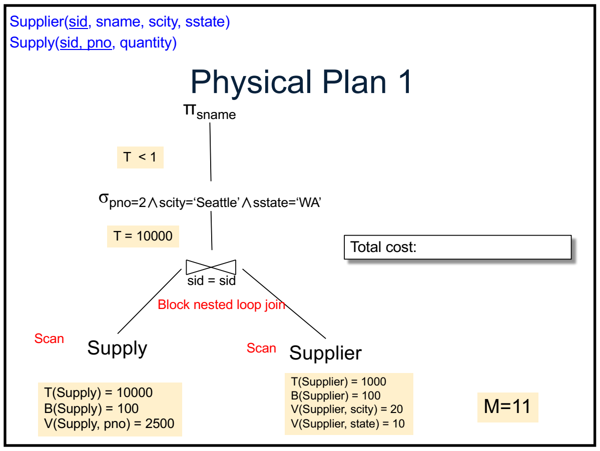


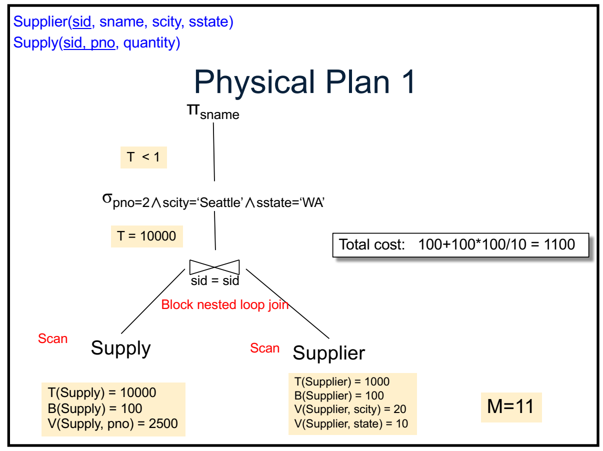


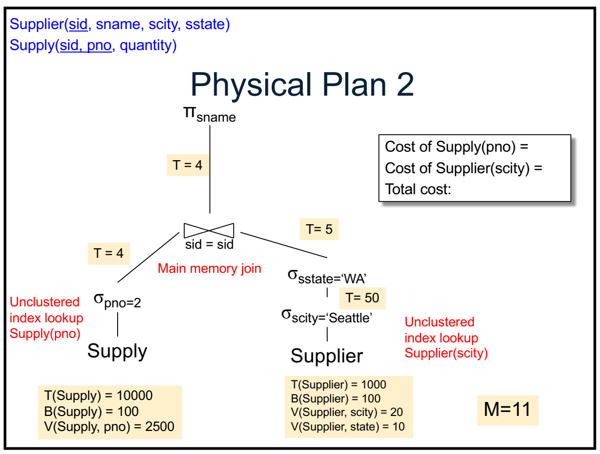


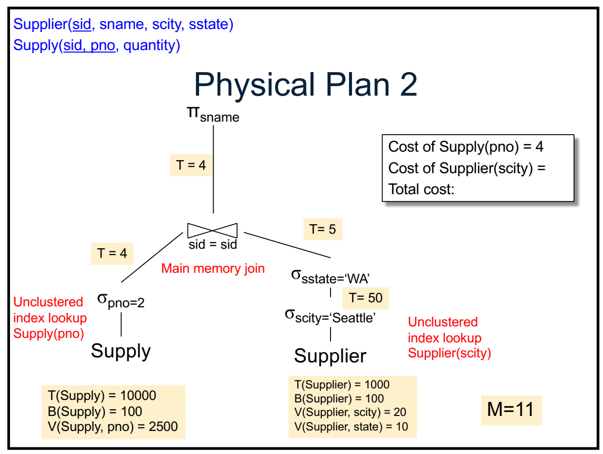


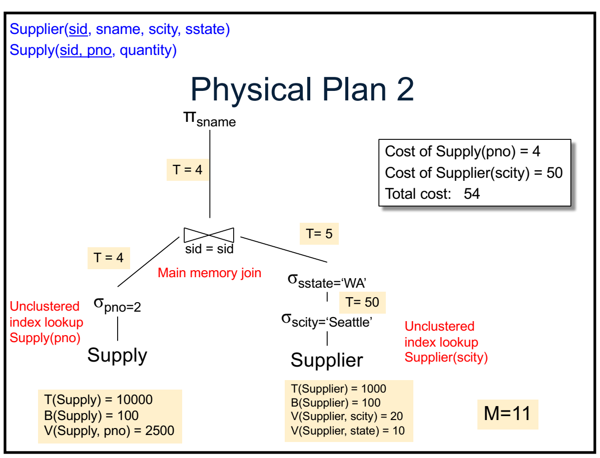
Computing Plan Costs

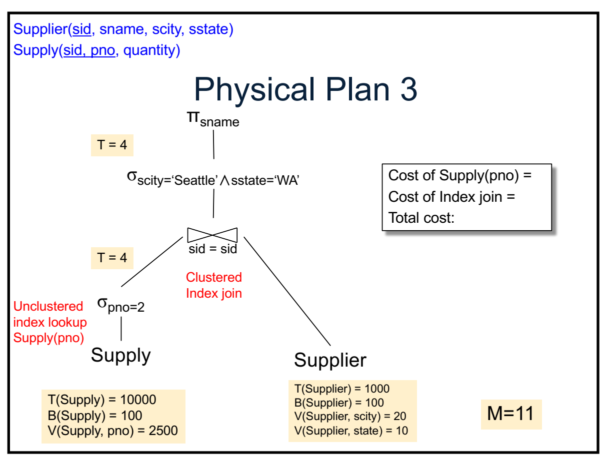

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 1
π sname
T < 1
σ pno=2∧scity='Seattle'∧sstate='WA'
T = 10000
Total cost:
sid = sid
Block nested loop join
Scan
Supply
Scan
Supplier
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 1
π sname
T < 1
σ pno=2∧scity='Seattle'∧sstate='WA'
T = 10000
Total cost: 100+100*100/10 = 1100
sid = sid
Block nested loop join
Scan
Supply
Scan
Supplier
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 2
π sname
Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:
T = 4
T = 4
sid = sid
T= 5
Main memory join
σ sstate='WA'
T= 50
σ scity='Seattle'
Unclustered index lookup Supply(pno)
σ pno=2
Supply
Supplier
Unclustered index lookup Supplier(scity)
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11

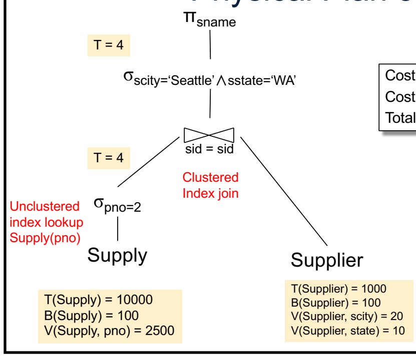

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 2
π sname
Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:
T = 4
T = 4
sid = sid
T= 5
Main memory join
σ sstate='WA'
T= 50
σ scity='Seattle'
Unclustered index lookup Supply(pno)
σ pno=2
Supply
Supplier
Unclustered index lookup Supplier(scity)
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 2
π sname
Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54
T = 4
T = 4
sid = sid
T= 5
Main memory join
σ sstate='WA'
T= 50
σ scity='Seattle'
Unclustered index lookup Supply(pno)
σ pno=2
Supply
Supplier
Unclustered index lookup Supplier(scity)
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 3
π sname
T = 4
σ scity='Seattle'∧sstate='WA'
Cost of Supply(pno) =
Cost of Index join =
Total cost:
T = 4
sid = sid
Clustered Index join
Unclustered index lookup Supply(pno)
σ pno=2
Supply
Supplier
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3

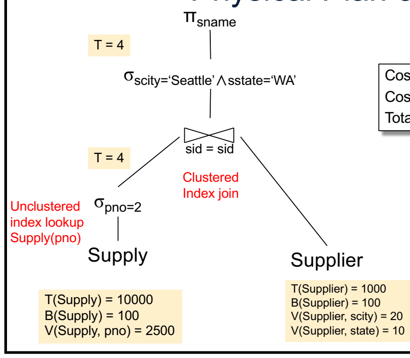


Cost of Supply(pno) = 4
Cost of Index join =
Total cost:

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

M=11

Query Optimizer Summary

- Input: A logical query plan
- Output: A good physical query plan
- Basic query optimization algorithm
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
  - Choose plan with lowest cost

- This is called cost-based optimization
  - More in CSE 444

CSE 414 - Spring 2018 45