## Introduction to Database Systems CSE 414

Lecture 17:
Basics of Query Optimization and Query Cost Estimation



## Announcements

- Midterm will be released by end of day today
- Need to start one HW6 step NOW:
  - https://aws.amazon.com/education/awseducate/apply/
  - Need to make an AWS account, can use existing Amazon account
  - Click on application button under Students and fill out form with your @uw.edu email
  - Will then be sent email for verification, must click to verify your email address



## Two typical kinds of queries

```
SELECT *
FROM Movie
WHERE year = ?
```

- Point queries
- What data structure should be used for index?

```
SELECT *
FROM Movie
WHERE year >= ? AND
      year <= ?
```

- Range queries
- What data structure should be used for index?



## Choosing Index is Not Enough

- To estimate the cost of a query plan, we still need to consider other factors:
  - How each operator is implemented
  - The cost of each operator
  - Let's start with the basics



## Cost of Reading Data From Disk



## Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

## Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**



## Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**

- DBMS collects statistics about base tables
  must infer them for intermediate results



One_year(did, month) e.g. (1, Jan), (2, Jan)…(365, Dec)

## Selectivity Factors for Conditions

How many tuples would this select:

SELECT *
FROM One_year
WHERE did = 32

1 tuple (out of 365)



One_year(did, month) e.g. (1, Jan), (2, Jan)…(365, Dec)

## Selectivity Factors for Conditions

How many tuples would this select:

SELECT *
FROM One_year
WHERE month = Jan

31 tuples (out of 365)
This is roughly 1/12 of the tuples, because 12 distinct values equally distributed.



## Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**

- DBMS collects statistics about base tables
  must infer them for intermediate results



## Selectivity Factors for Conditions

- A = c /* $\sigma_{A=c}$ (R) */
  - Selectivity f = 1/V(R,A)
- A < c /* $\sigma_{A<c}$ (R)*/
  - Selectivity f = (c - min(R, A))/(max(R,A) - min(R,A))
- c1 < A < c2 /* $\sigma_{c1<A<c2}$ (R)*/
  - Selectivity f = (c2 – c1)/(max(R,A) - min(R,A))
- Cond1 ∧ Cond2 ∧ Cond3 ∧ ... –
  - Selectivity = f1*f2*f3*...(assumes independence)

## Cost of Reading Data From Disk

- Sequential scan for relation R costs **B(R)**
- Index-based selection
  - Estimate selectivity factor **f** (see previous slide)
  - Clustered index: f***B(R)**
  - Unclustered index f***T(R)**

Note: we ignore I/O cost for index pages



## Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R)$ = ?
- Table scan:
- Index based selection:



## Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R)$ = ?
- Table scan: B(R) = 2,000 I/Os
- Index based selection:



## Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R)$ = ?
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered:
  - If index is unclustered:



## Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R)$ = ?
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
    Why: we know we can scan a full block to get the desired range
  - If index is unclustered:



## Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R)$ = ?
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
    Why: we know we can scan a full block to get the desired range
  - If index is unclustered: T(R) * 1/V(R,a) = 5,000 I/Os

## Index Based Selection

- Example:

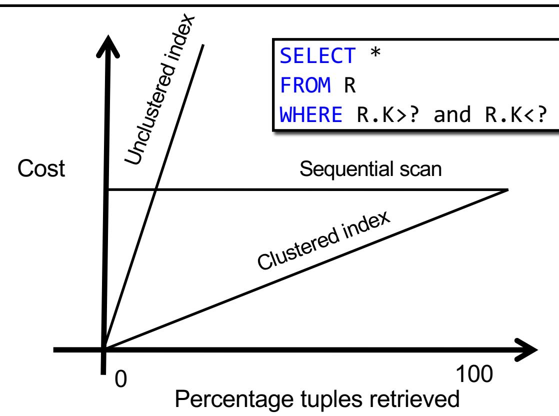
| |
|---|
| B(R) = 2000 |
| T(R) = 100,000 |
| V(R, a) = 20 |

cost of $\sigma_{a=v}(R) = ?$

- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
    Why: we know we can scan a full block to get the desired range
  - If index is unclustered: T(R) * 1/V(R,a) = 5,000 I/Os

Lesson: Don't build unclustered indexes when V(R,a) is small !








## Cost of Executing Operators (Focus on Joins)



## Outline

- **Join operator algorithms**
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
- Note about readings:
  - In class, we discuss only algorithms for joins
  - Other operators are easier: read the book



## Join Algorithms

- Hash join
- Nested loop join



## Hash Join

Hash join: $R \bowtie S$

- Scan R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)
- Which relation to build the hash table on?

# Hash Join

Hash join: R ⋈ S

- Scan R, build buckets 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

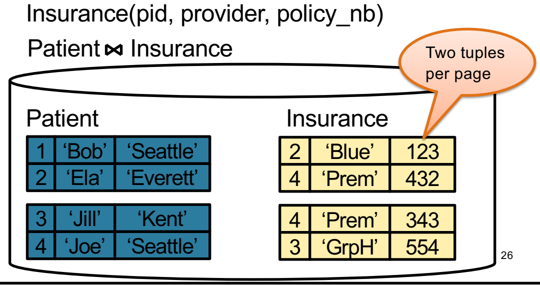


# Hash Join Example

Patient(pid, name, address)

Insurance(pid, provider, policy_nb)

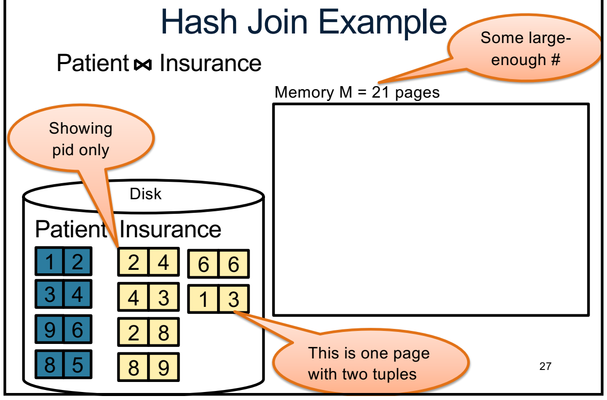
Patient ⋈ Insurance






# Hash Join Example

Patient ⋈ Insurance

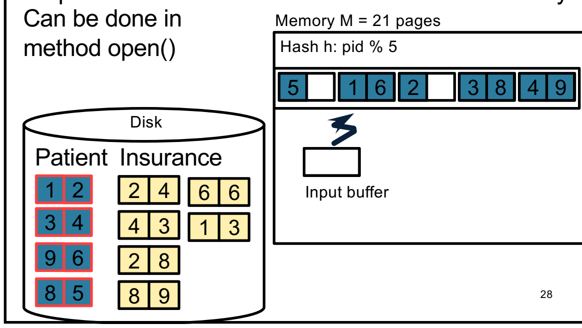





# Hash Join Example

Step 1: Scan Patient and build hash table in memory

Can be done in method open()

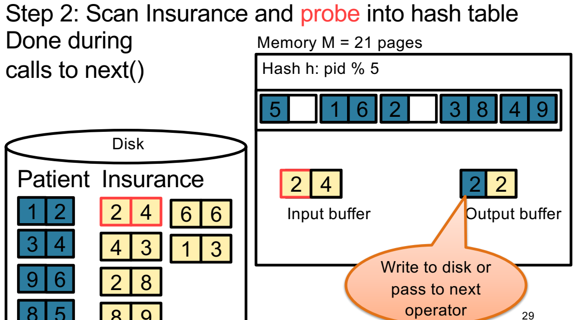





# Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

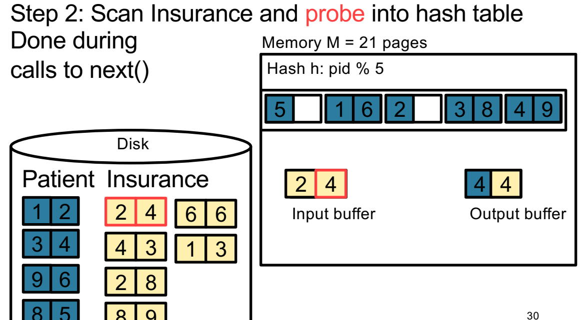





# Hash Join Example

Step 2: Scan Insurance and probe into hash table

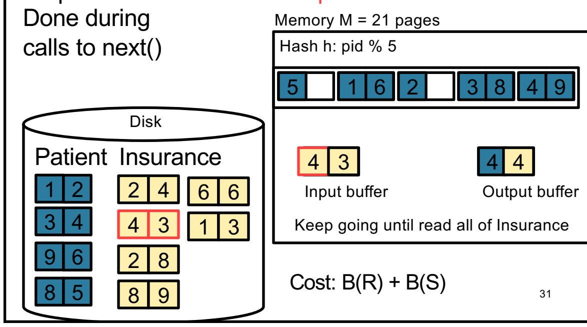
Done during calls to next()



# Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during calls to next()




Cost: B(R) + B(S)



# Nested Loop Joins

- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

for each tuple $t_1$ in R do
  for each tuple $t_2$ in S do
    if $t_1$ and $t_2$ join then output ($t_1$,$t_2$)

What is the Cost?



# Nested Loop Joins

- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

for each tuple $t_1$ in R do
  for each tuple $t_2$ in S do
    if $t_1$ and $t_2$ join then output ($t_1$,$t_2$)

What is the Cost?

- Cost: B(R) + T(R) B(S)
- Multiple-pass since S is read many times



# Page-at-a-time Refinement

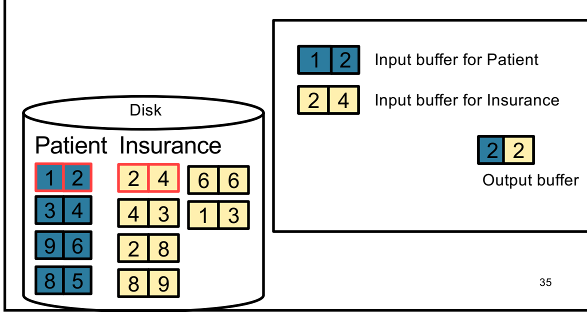
for each page of tuples r in R do
  for each page of tuples s in S do
    for all pairs of tuples $t_1$ in r, $t_2$ in s
      if $t_1$ and $t_2$ join then output ($t_1$,$t_2$)

- Cost: B(R) + B(R)B(S)

What is the Cost?

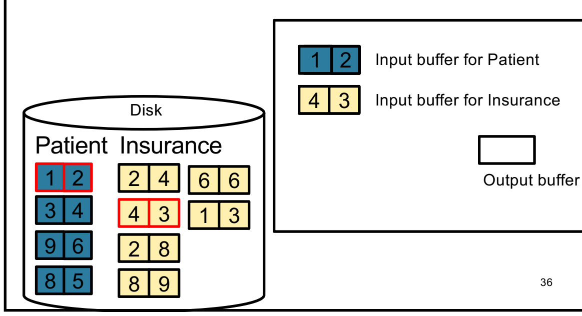


# Page-at-a-time Refinement

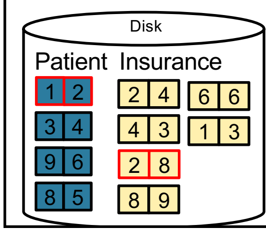


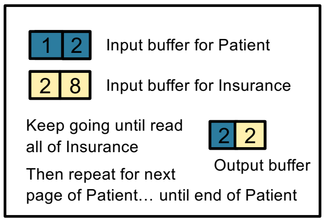



# Page-at-a-time Refinement



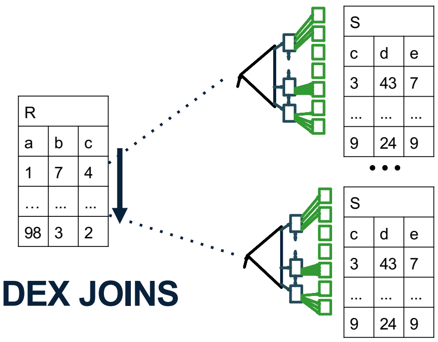
# Page-at-a-time Refinement




Cost: B(R) + B(R)B(S)






# 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
- 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))



# Index Nested Loop Join

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

Still have to scan in R

Why is the multiplier term T(R)?

What does 1/V(S,a) represent?

T(R) must be used because we cannot assume that a whole block of R (B(R)) will have the same attribute to join on, and thus use the same index access on S for.

1/V(S,a) represents the nature of the B+ Tree index. We are only scanning as much as we need. Note that the performance of the index join will decrease as V decreases.



# Index Nested Loop Join

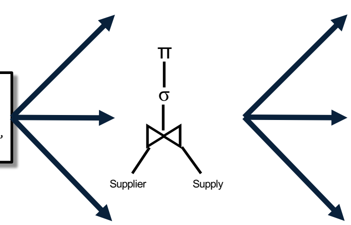
If index on S is unclustered:
B(R) + T(R) * (T(S) * 1/V(S,a))

Why did this change from B(R) to T(R)?

Remember that tuples are stored on contiguous blocks. In a clustered index from before we know we can scan a single chunk of the disk to get the entire desired range. In an unclustered index we no longer can assume contiguous access. Thus we estimate that every tuple needs its own I/O operation.



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




# GENERATING QUERY PLANS (REVIEW)

## Review: Logical vs Physical Plans

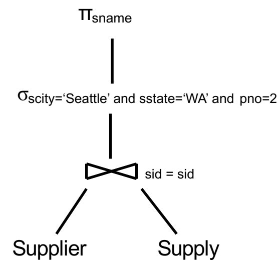
- Logical plans:
  – Created by the parser from the input SQL text
  – Expressed as a relational algebra tree
  – Each SQL query has many possible logical plans
- Physical plans:
  – Goal is to choose an efficient implementation for each operator in the RA tree
  – Each logical plan has many possible physical plans



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

## Review: Relational Algebra

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

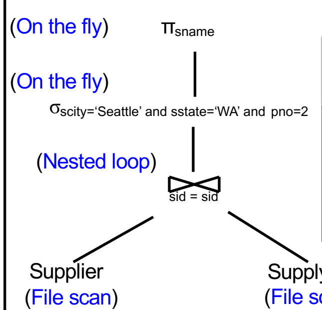



Relational algebra expression is also called the "logical query plan"



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

## Review: Physical Query Plan 1



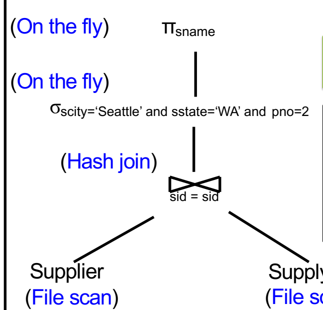

A physical query plan is a logical query plan annotated with physical implementation details

```
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(sid, sname, scity, sstate)
Supply(sid, pno, 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'
```



## Query Optimization: Overview

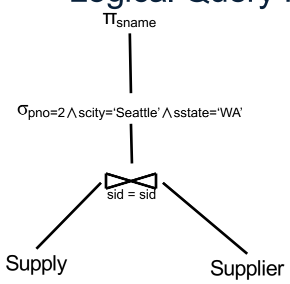
- Compute cost of each operator
  – This depends on:
    - Table statistics (# of tuples etc)
    - Algorithm used
- Cost of a physical plan = sum(each operator cost)
- Cost each plan and choose the one with lowest cost



## Cost of Query Plans

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'
```

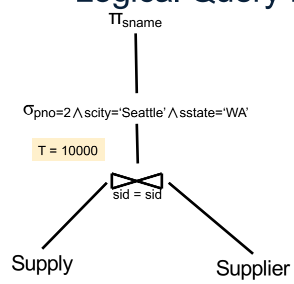
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10



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'
```

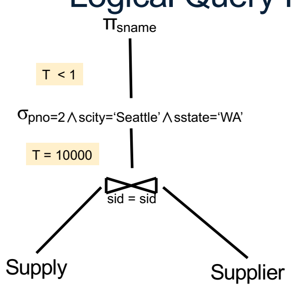
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10



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'
```

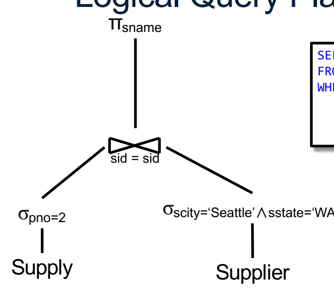
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10



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

## Logical Query Plan 2




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

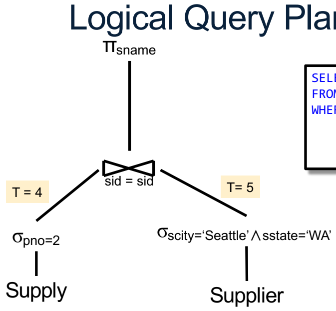
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10



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

## Logical Query Plan 2




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

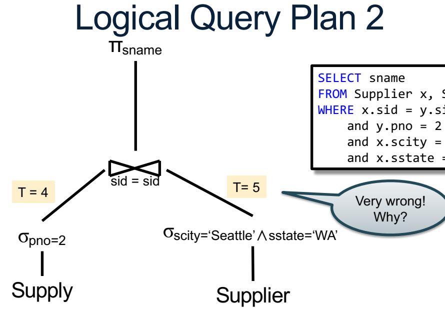
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10



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

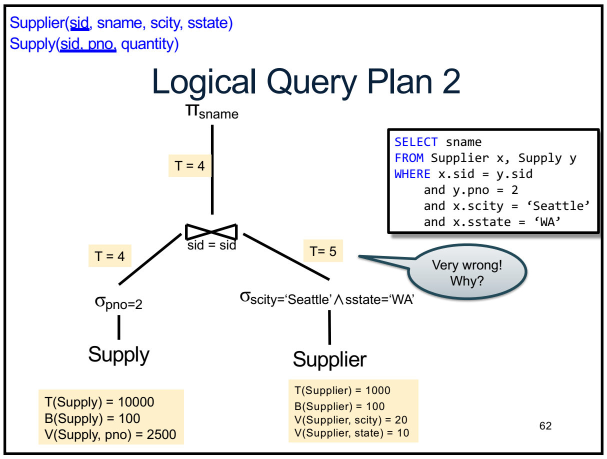
## Logical Query Plan 2

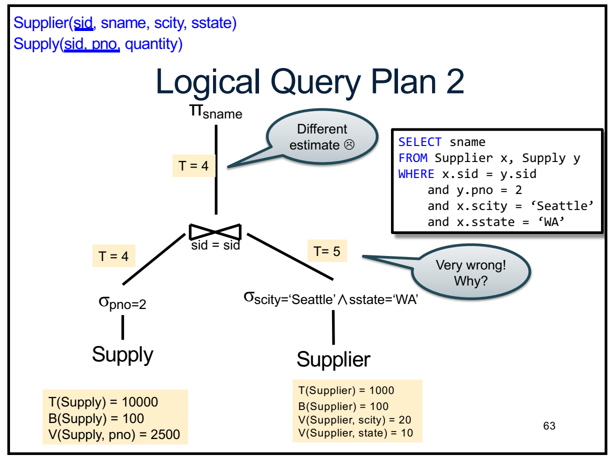


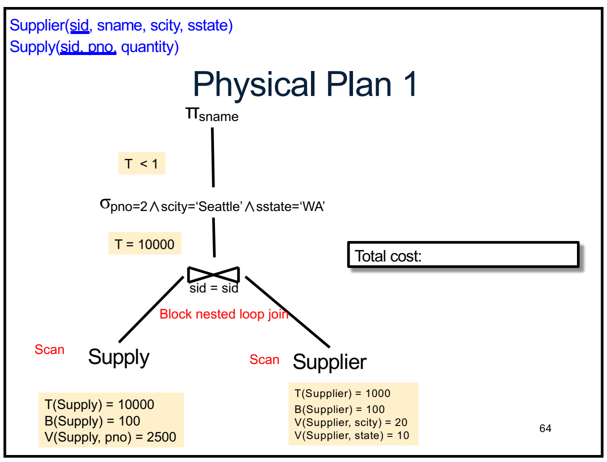

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

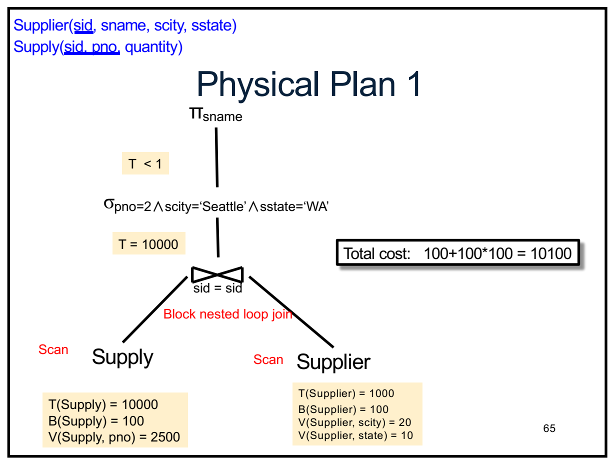
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

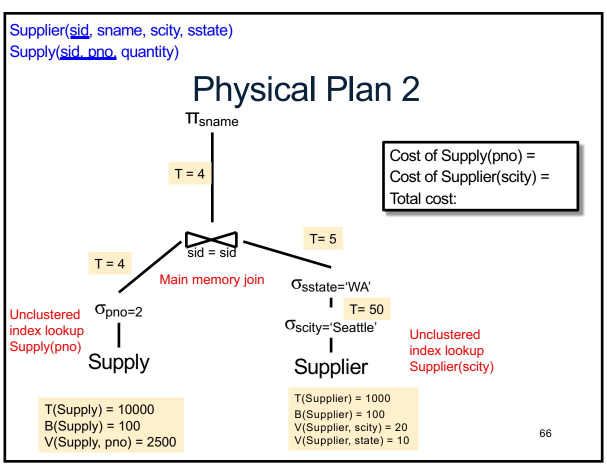
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

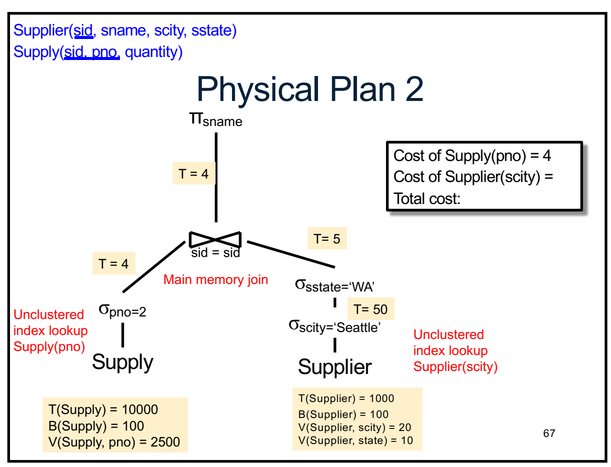

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Logical Query Plan 2
πsname
T = 4
sid = sid
T = 4
T= 5
σpno=2
σscity='Seattle'∧sstate='WA'
Supply
Supplier
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
and y.pno = 2
and x.scity = 'Seattle'
and x.sstate = 'WA'
Very wrong! Why?
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
62

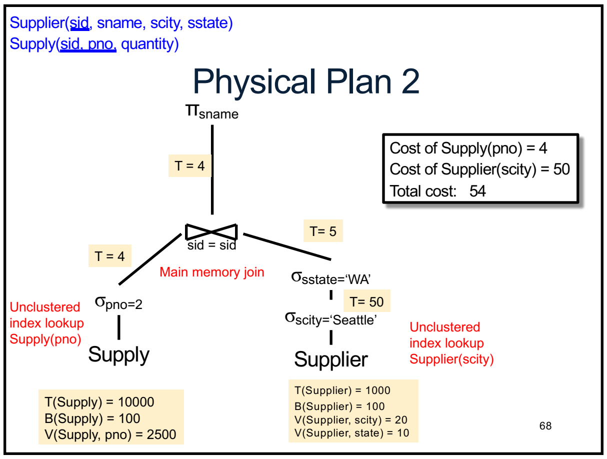

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Logical Query Plan 2
πsname
Different estimate ☹
T = 4
sid = sid
T = 4
T= 5
σpno=2
σscity='Seattle'∧sstate='WA'
Supply
Supplier
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
and y.pno = 2
and x.scity = 'Seattle'
and x.sstate = 'WA'
Very wrong! Why?
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
63

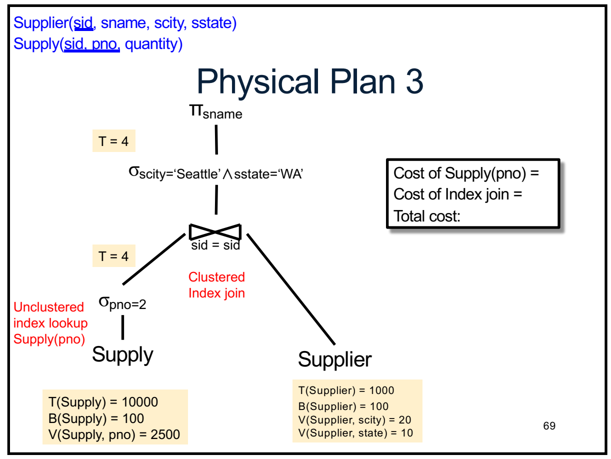

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 1
πsname
T < 1
σpno=2∧scity='Seattle'∧sstate='WA'
T = 10000
Total cost:
sid = sid
Block nested loop join
Scan
Supply
Scan
Supplier
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
64

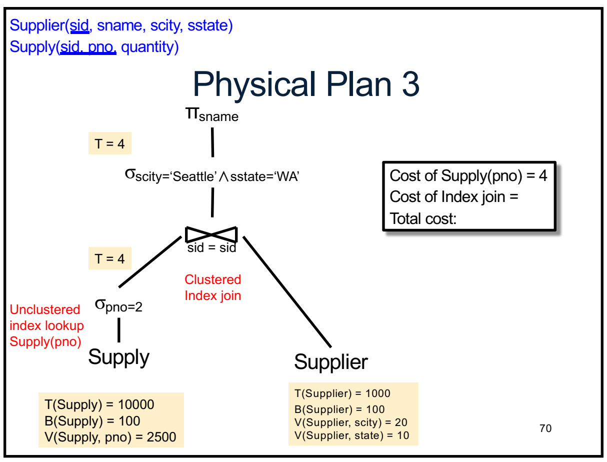

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 1
πsname
T < 1
σpno=2∧scity='Seattle'∧sstate='WA'
T = 10000
Total cost: 100+100*100 = 10100
sid = sid
Block nested loop join
Scan
Supply
Scan
Supplier
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
65

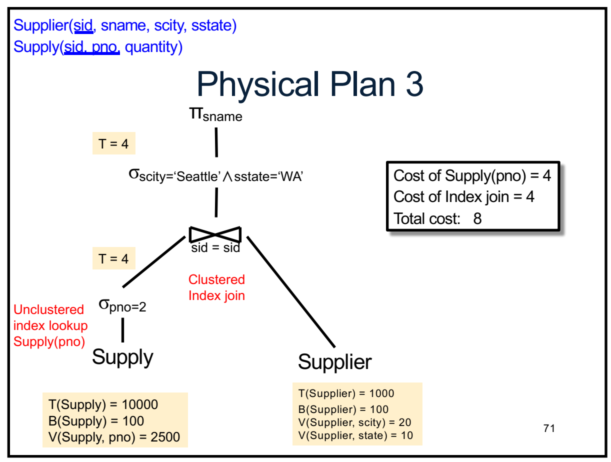

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 2
πsname
T = 4
Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:
sid = sid
T = 4
T= 5
Main memory join
σsstate='WA'
T= 50
Unclustered index lookup Supply(pno)
σpno=2
σscity='Seattle'
Supply
Supplier
Unclustered index 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
66


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Physical Plan 2
πsname
T = 4
Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:
sid = sid
T = 4
T= 5
Main memory join
σsstate='WA'
T= 50
Unclustered index lookup Supply(pno)
σpno=2
σscity='Seattle'
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
67




## 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