Database Systems
CSE 414

Lectures 11 – 12:
Basics of Query Optimization and Cost Estimation
(Ch. 15.{1,3,4.6,6} & 16.4-5)

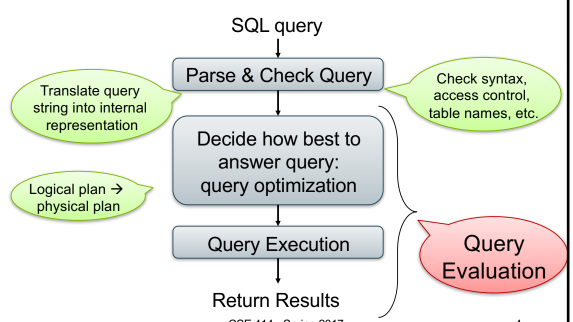
Announcements

- HW3 is due Tuesday
- WQ4 is due Thursday
- Midterm on Friday
  – we'll talk more about it on Monday
- Husky Football spring game tomorrow

Motivation

- To understand performance, need to understand a bit about how a DBMS works
  – my database application is too slow… why?
  – one of the queries is very slow… why?
- Under your direct control: index choice
  – understand how that affects query performance

Recap: Query Evaluation



Query Optimizer Overview

- **Input**: Parsed & checked SQL
- **Output**: A good physical query plan
- **Basic query optimization algorithm:**
  – Enumerate alternative plans (logical and physical)
  – Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  – Choose plan with lowest cost
  – This is called cost-based optimization

Query Optimizer Overview

- There are exponentially many query plans
  – exponential in the size of the query
  – simple SFW with 3 joins has not too many
- Optimizer will consider many, many of them
- Worth substantial cost to avoid **bad plans**

Rest of Today

- Cost of reading from disk
- Cost of single RA operators
- Cost of query plans

Cost of Reading Data From Disk

Cost Parameters

- Cost = Disk I/O + CPU + Network I/O
  - We will focus on Disk I/O
- Parameters:
  - **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)**
- Where do these values come from?
  - DBMS collects statistics about data on disk

Selectivity Factors for Conditions

- A = c /* $\sigma_{A=c}(R)$ */
  - Selectivity = 1/V(R,A)
- A < c /* $\sigma_{A<c}(R)$*/
  - Selectivity = (c - Low(R, A))/(High(R,A) - Low(R,A))
- c1 < A < c2 /* $\sigma_{c1<A<c2}(R)$*/
  - Selectivity = (c2 – c1)/(High(R,A) - Low(R,A))

Example: Selectivity of $\sigma_{A=c}(R)$

T(R) = 100,000
V(R, A) = 20

How many records are returned by $\sigma_{A=c}(R)$ = ?

Answer: X * T(R), where X = selectivity…
... X = 1/V(R,A) = 1/20

Number of records returned = 100,000/20 = 5,000

Cost of Index-based Selection

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

Note: we are ignoring I/O cost for index pages

Example: Cost of $\sigma_{A=c}(R)$

- Example:

B(R) = 2000
T(R) = 100,000
V(R, A) = 20

cost of $\sigma_{A=c}(R)$ = ?

- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R)/V(R,A) = 100 I/Os
  - If index is unclustered: T(R)/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
- Sort-merge join

Hash Join

Hash join: R ⋈ S

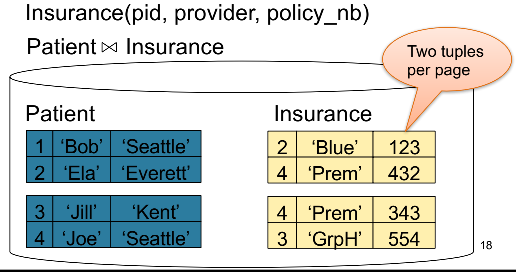
- Scan R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)
- One-pass algorithm when B(R) ≤ M
  - more disk access also when B(R) > M

Hash Join Example

Patient(pid, name, address)

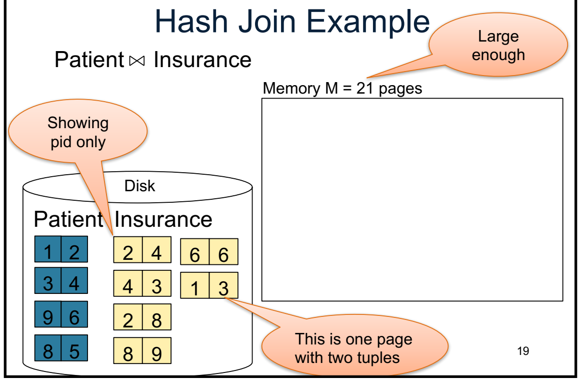
Insurance(pid, provider, policy_nb)

Patient ⋈ Insurance



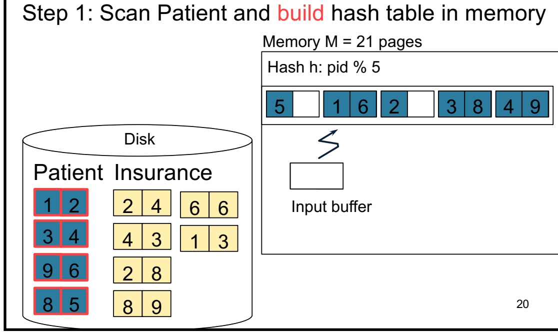
Hash Join Example

Patient ⋈ Insurance



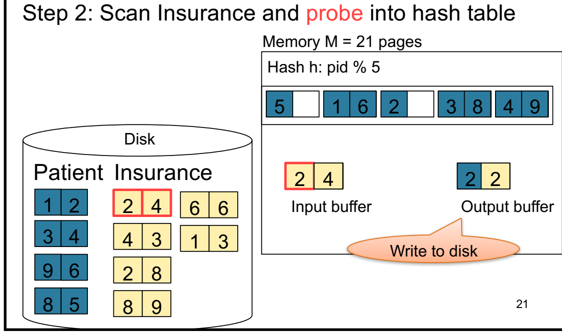
Hash Join Example

Step 1: Scan Patient and build hash table in memory



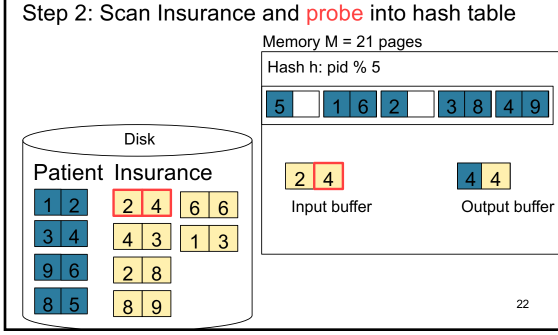
Hash Join Example

Step 2: Scan Insurance and probe into hash table



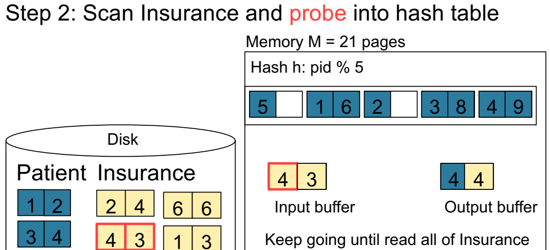
Hash Join Example

Step 2: Scan Insurance and probe into hash table



Hash Join Example

Step 2: Scan Insurance and probe into hash table



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 t1 in R do
   for each tuple t2 in S do
      if t1 and t2 join then output (t1,t2)
```

What is the Cost?

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

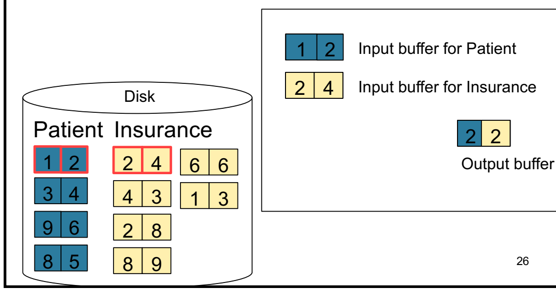
Block-at-a-time Refinement

for each block of tuples r in R do
  for each block 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)

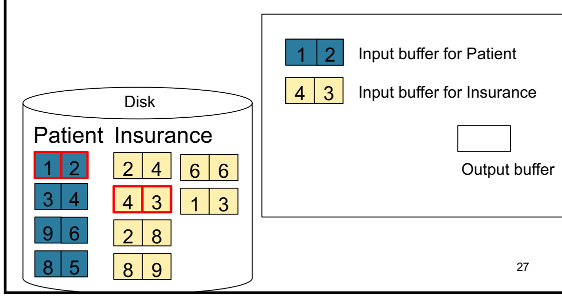
- Cost: B(R) + B(R)B(S)

What is the Cost?

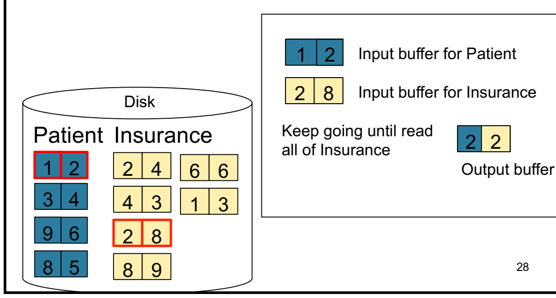
Block-at-a-time Refinement



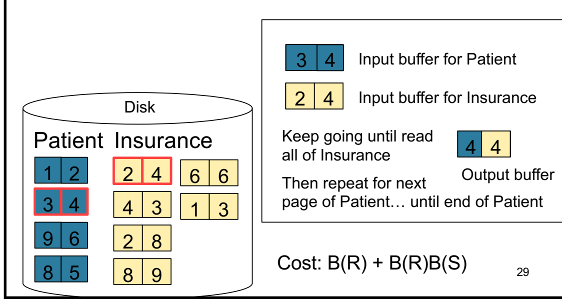
Block-at-a-time Refinement



Page-at-a-time Refinement



Block-at-a-time Refinement



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

Block-Nested-Loop Refinement

for each group of M-1 pages 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)/(M-1)

What is the Cost?

Sort-Merge Join

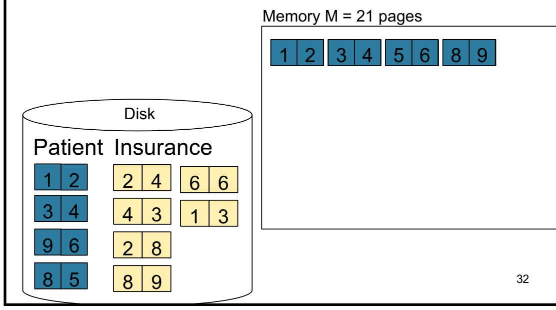
Sort-merge join: R ⋈ S

- 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

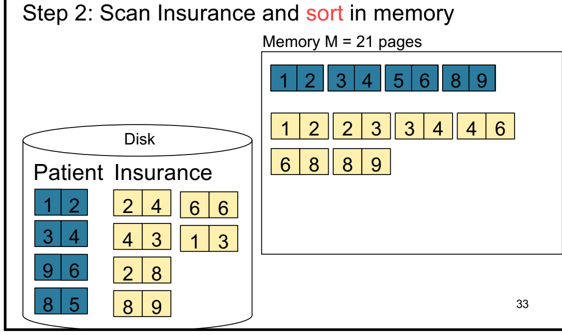
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory



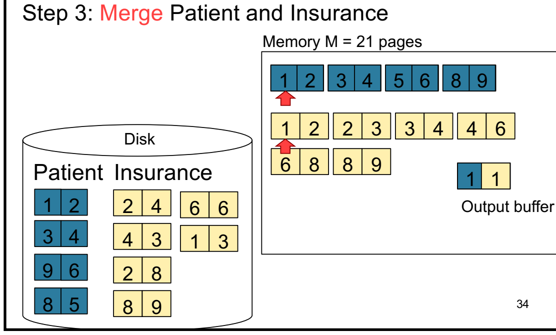
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory



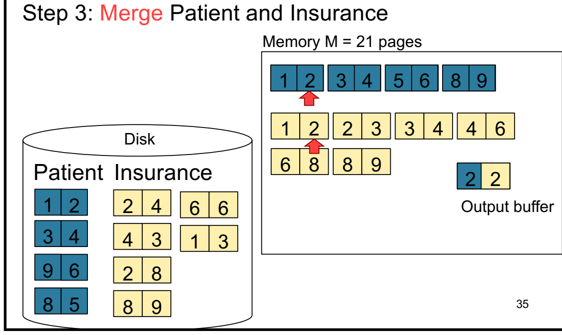
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



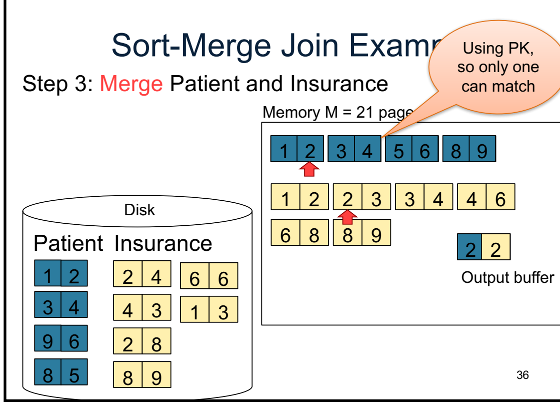
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



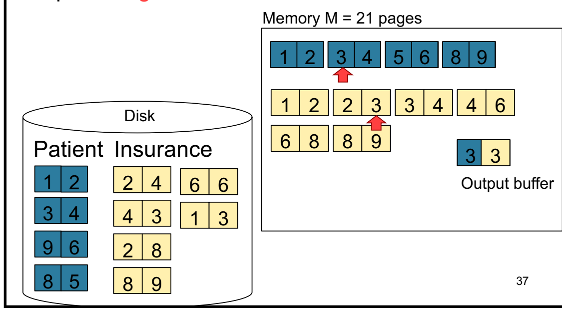
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



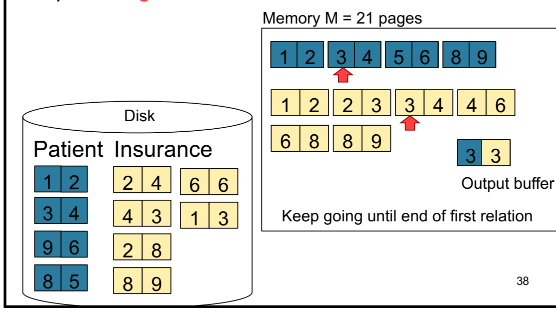
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Index Nested Loop Join

R ⋈ S

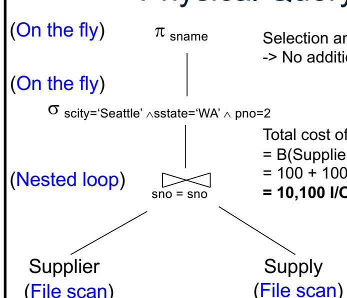
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

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

Cost of Query Plans

T(Supplier) = 1000
T(Supply) = 10,000
B(Supplier) = 100
B(Supply) = 100
V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500
M = 11

Physical Query Plan 1

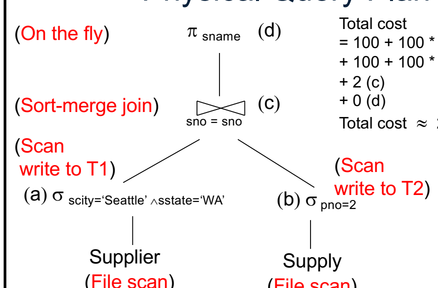


Selection and project on-the-fly
-> No additional cost.

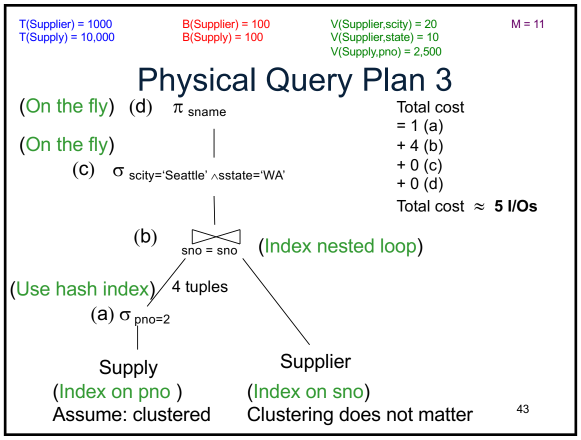
Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)
= 100 + 100 * 100
= 10,100 I/Os

T(Supplier) = 1000
T(Supply) = 10,000
B(Supplier) = 100
B(Supply) = 100
V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500
M = 11

Physical Query Plan 2



Total cost
= 100 + 100 * 1/20 * 1/10 (a)
+ 100 + 100 * 1/2500 (b)
+ 2 (c)
+ 0 (d)
Total cost ≈ **204 I/Os**


T(Supplier) = 1000
T(Supply) = 10,000
B(Supplier) = 100
B(Supply) = 100
V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500
M = 11
Physical Query Plan 3
(On the fly) (d) π sname
(On the fly)
(c) σ scity='Seattle' ∧sstate='WA'
(b) ⋈ sno = sno
(Index nested loop)
(Use hash index)
4 tuples
(a) σ pno=2
Supply
(Index on pno)
Assume: clustered
Supplier
(Index on sno)
Clustering does not matter
Total cost
= 1 (a)
+ 4 (b)
+ 0 (c)
+ 0 (d)
Total cost ≈ 5 I/Os
43