Query Optimization

Introduction to Databases

CompSci 316 Spring 2017

DUKE COMPUTER SCIENCE

Announcements (Wed., Mar. 29)

- Homework #3
  - 3.3, 3.4, and 3.5 due today

Review QP and QO

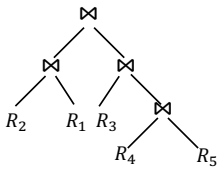
- QO approaches
  - Cost-based QO
  - Heuristic (rule)-based QO
  - Query rewriting, unnesting, and decorrelation
- Equivalence of RA operators
- Estimate cost of individual operators (Lec 16 + 17)
- Estimate output size of individual operators (Lec 18)
  - Uniformity + Independence + Other assumptions
  - Selection
  - Join
- Today:
  - Explore search space for join only plans
  - Combine multiple operators

Search strategy

http://1.bp.blogspot.com/-MoIduSreRKs/TgyAi4ki5QI/AAAAAAAAAKE/mi8ejfZ8S7U/s1600/cornMaze.jpg

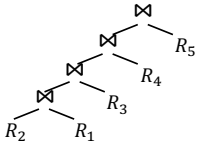
Search space

- Huge!
- "Bushy" plan example:



- Just considering different join orders, there are $\frac{(2n-2)!}{(n-1)!}$ bushy plans for $R_1 \bowtie \cdots \bowtie R_n$
  - 30240 for $n = 6$
    - (FYI: Recurrence relation: f(n)=f(n-1)*(4n-6), where 4n-6 comes from two ways to add the new node to one of n-1 leaves and n-2 internal nodes.)
- And there are more if we consider:
  - Multiway joins
  - Different join methods
  - Placement of selection and projection operators

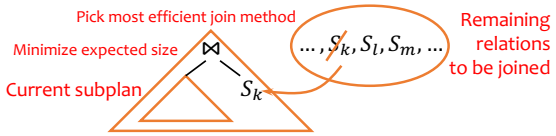
Left-deep plans



- Heuristic: consider only "left-deep" plans, in which only the left child can be a join
  - Tend to be better than plans of other shapes, because many join algorithms scan inner (right) relation multiple times—you will not want it to be a complex subtree
- How many left-deep plans are there for $R_1 \bowtie \cdots \bowtie R_n$?
  - Significantly fewer, but still lots—$n!$ (720 for $n = 6$)

A greedy algorithm

- $S_1, \ldots, S_n$
  - Say selections have been pushed down; i.e., $S_i = \sigma_p(R_i)$
- Start with the pair S_i, S_j with the smallest estimated size for $S_i \bowtie S_j$
- Repeat until no relation is left:
  Pick S_k from the remaining relations such that the join of S_k and the current result yields an intermediate result of the smallest size

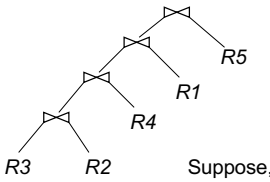


A dynamic programming approach

- Selinger's algorithm
  - IBM System R, frequently adapted and used
- Generate optimal plans bottom-up
  - Pass 1: Find the best single-table plans (for each table)
  - Pass 2: Find the best two-table plans (for each pair of tables) by combining best single-table plans
  - …
  - Pass k: Find the best k-table plans (for each combination of k tables) by combining two smaller best plans found in previous passes
  - …
- Rationale: Any subplan of an optimal plan must also be optimal
  - otherwise, just replace the subplan to get a better overall plan

Principle of Optimality

Query: *R1* ⋈ *R2* ⋈ *R3* ⋈ *R4* ⋈ *R5*

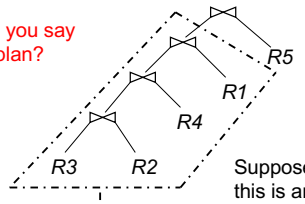


Suppose,
this is an Optimal Plan
for joining R1…R5:

Principle of Optimality

Query: *R1* ⋈ *R2* ⋈ *R3* ⋈ *R4* ⋈ *R5*

Then, what can you say about this sub-plan?



Suppose,
this is an Optimal Plan
for joining R1…R5:

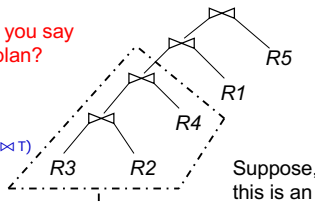
This has to be the optimal plan for joining *R3, R2, R4, R1*

Principle of Optimality

Query: *R1* ⋈ *R2* ⋈ *R3* ⋈ *R4* ⋈ *R5*

Then, what can you say about this sub-plan?

We are using the associativity and commutativity of joins
(R ⋈ S) ⋈ T = R ⋈ (S ⋈ T)
R ⋈ S = S ⋈ R



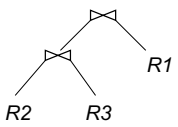
Suppose,
this is an Optimal Plan
for joining R1…R5:

This has to be the optimal plan for joining *R3, R2, R4*

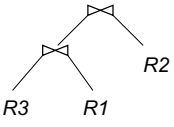
Exploiting Principle of Optimality

Query: *R1* ⋈ *R2* ⋈ … ⋈ *Rn*

Both are giving the same result
R2 ⋈ R3 ⋈ R1 = R3 ⋈ R1 ⋈ R2



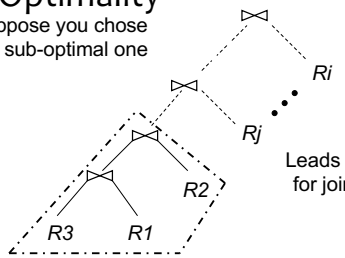
Optimal
for joining *R1, R2, R3*



Sub-Optimal
for joining *R1, R2, R3*

Exploiting Principle of Optimality

Suppose you chose the sub-optimal one



Leads to sub-Optimal for joining R1,…,Rn

A sub-optimal sub-plan cannot lead to an optimal plan

13

Selinger Algorithm:

OPT ({ R1, R2, R3 }):

Min
- OPT ({ R1, R2 }) + Join (({R1, R2}), R3)
- OPT ({ R2, R3 }) + Join (({R2, R3}), R1)
- OPT ({ R1, R3 }) + Join (({R3, R1}), R2)}

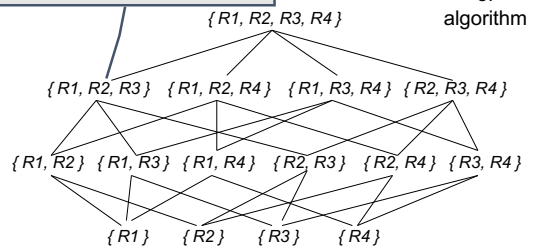
14

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

e.g. All possible permutations of R1, R2, R3 have been considered after OPT({R1, R2, R3}) has been computed

Progress of algorithm

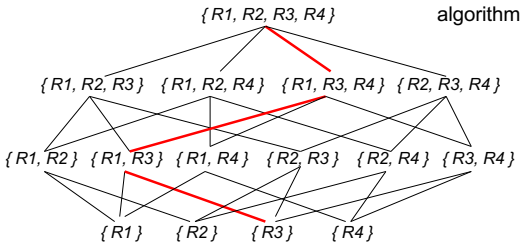


15

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

Progress of algorithm



16

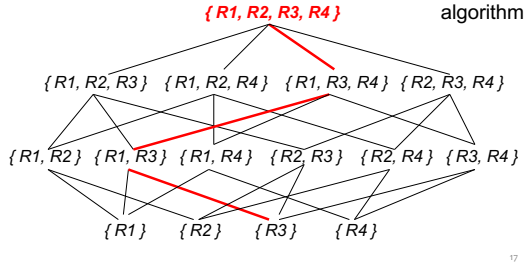
Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

Q. How to optimally compute join of {R1, R2, R3, R4}?

Ans: First optimally join {R1, R3, R4} then join with R2 as inner.

Progress of algorithm



17

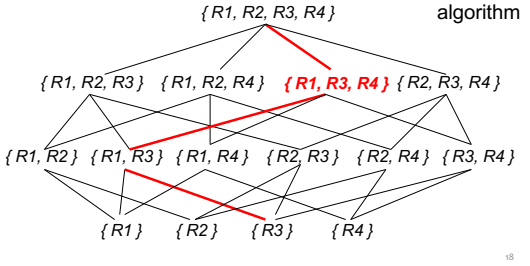
Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

Q. How to optimally compute join of {R1, R3, R4}?

Ans: First optimally join {R1, R3}, then join with R4 as inner.

Progress of algorithm



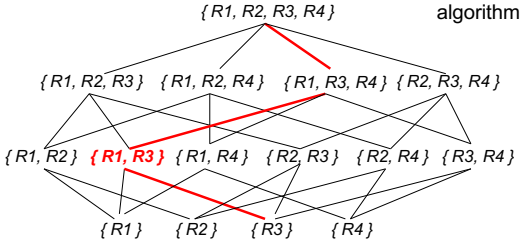
18

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

Q. How to optimally compute join of {R1, R3}?

Ans: First optimally join {R3}, then join with R1 as inner.



Progress of algorithm

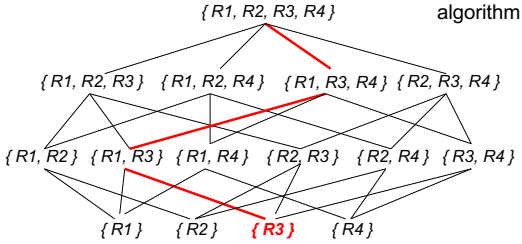
19

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

Q. How to optimally compute join of {R3}?

Ans: Single relation – so optimally scan R3.



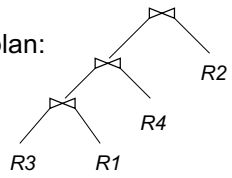
Progress of algorithm

20

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

Final optimal plan:



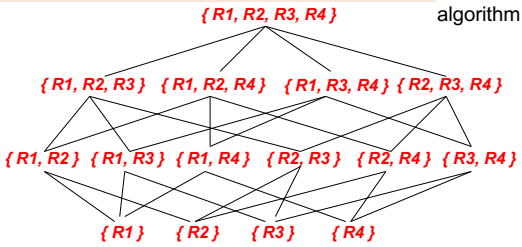
NOTE : There is a one-one correspondence between the permutation (R3, R1, R4, R2) and the above left deep plan

21

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$

NOTE: (*VERY IMPORTANT*)
- This is *NOT* done by top-down recursive calls.
- This is done BOTTOM-UP computing the optimal cost of *all* nodes in this lattice only once (dynamic programming).



Progress of algorithm

22

Summary: A dynamic programming approach

23

- Selinger's algorithm
- Generate optimal plans bottom-up
- Rationale: Any subplan of an optimal plan must also be optimal (otherwise, just replace the subplan to get a better overall plan)

☞Well, not quite...
- for physical plan

The need for "interesting order"

24

- Example: $R(A,B) \bowtie S(A,C) \bowtie T(A,D)$
- Best plan for $R \bowtie S$: hash join (beats sort-merge join)
- Best overall plan: sort-merge join R and S, and then sort-merge join with T
  - Subplan of the optimal plan is not optimal!
- Why?
  - The result of the sort-merge join of R and S is sorted on A
  - This is an interesting order that can be exploited by later processing (e.g., join, dup elimination, GROUP BY, ORDER BY, etc.)!

25

Dealing with interesting orders

When picking the best plan

- Comparing their costs is not enough
  - Plans are not totally ordered by cost anymore
- Comparing interesting orders is also needed
  - Plans are now partially ordered
  - Plan X is better than plan Y if
    - Cost of X is lower than Y, and
    - Interesting orders produced by X "subsume" those produced by Y
- Need to keep a set of optimal plans for joining every combination of k tables
  - At most one for each interesting order

26

Combine cost of different operators in a plan

- Given a Physical Plan
- Size = #tuples, NOT #blocks
- Cost = #page or block I/O
  - but, need to consider whether the intermediate relation fits in memory, is written back to disk (or on-the-fly goes to the next operator) etc.

Acknowledgement: Adapted from slides by Profs. Magda Balazinska and Dan Suciu

27

Example Query

Student (sid, name, age, address)
Book(bid, title, author)
Checkout(sid, bid, date)

Query:

```
SELECT S.name
FROM Student S, Book B, Checkout C
WHERE S.sid = C.sid
AND B.bid = C.bid
AND B.author = 'Olden Fames'
AND S.age > 12
AND S.age < 20
```

28

Assumptions

S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

- Student: S, Book: B, Checkout: C

- Sid, bid foreign key in C referencing S and B resp.
- There are 10,000 Student records stored on 1,000 blocks.
- There are 50,000 Book records stored on 5,000 blocks.
- There are 300,000 Checkout records stored on 15,000 blocks.
- There are 500 different authors.
- Student ages range from 7 to 24.

- **T(R) = no. of tuples**
- **B(R) = no. of blocks of R**
- **V(R, A) = no. of distinct values of attributes A of R**

29

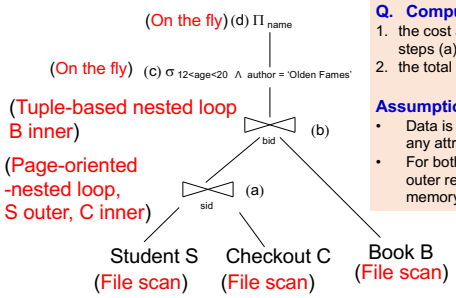
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

Physical Query Plan – 1



Q. Compute
1. the cost and cardinality in steps (a) to (d)
2. the total cost

Assumptions:
- Data is not sorted on any attributes
- For both in (a) and (b), outer relations fit in memory

30

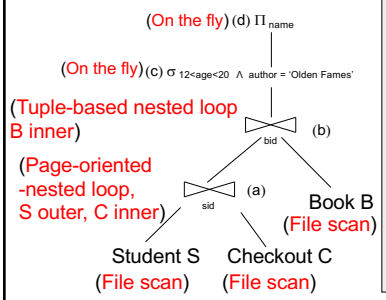
S(sid,name,age,addr)
B(bid,title,author)
C(sid,bid,date)

T(S)=10,000
T(B)=50,000
T(C)=300,000

B(S)=1,000
B(B)=5,000
B(C)=15,000

V(B,author) = 500
7 <= age <= 24

(a)



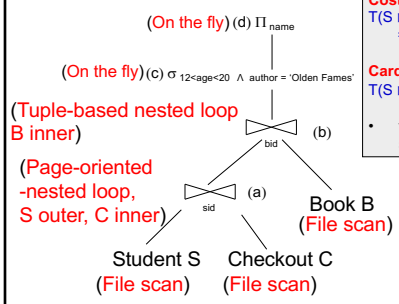
Cost =
B(S) + B(S) * B(C)
= 1000 + 1000 * 15000
= 15,001,000

Cardinality =
T(C) = 300,000
- foreign key join, output pipelined to next join
- Can apply the formula as well

T(S) * T(C)/max (V(S, sid), V(C, sid))
= T(C)
since V(S, sid) > = V(C, sid)
and
T(S) = V(S, sid)

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author) T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date) T(C)=300,000 B(C)=15,000

(b)



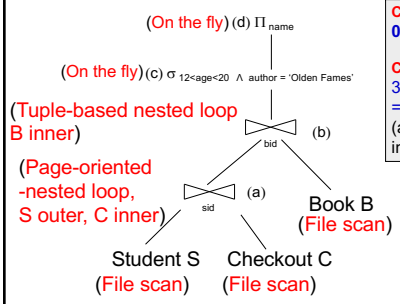
Cost =
T(S ⋈ C) * B(B)
= 300,000 * 5,000 = 15 * 10^8

Cardinality =
T(S ⋈ C) = 300,000

- foreign key join, don't need scanning for outer relation

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author) T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date) T(C)=300,000 B(C)=15,000

(c, d)

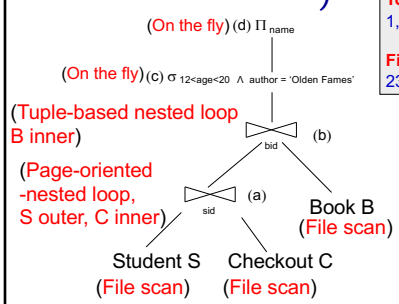


Cost =
0 (on the fly)

Cardinality =
300,000 * 1/500 * 7/18
= 234 (approx)
(assuming uniformity and independence)

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author) T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date) T(C)=300,000 B(C)=15,000

(Total
)

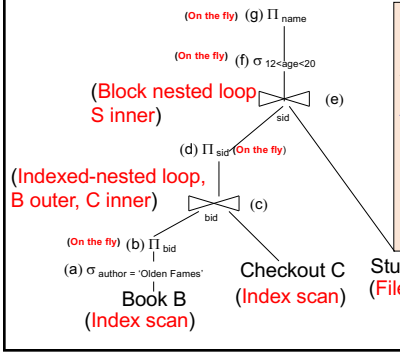


Total cost =
1,515,001,000

Final cardinality =
234 (approx)

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author) T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date) T(C)=300,000 B(C)=15,000

Physical Query Plan – 2

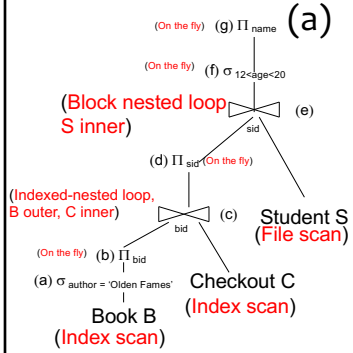


Q. Compute
1. the cost and cardinality in steps (a) to (g)
2. the total cost

Assumptions:
- Unclustered B+tree index on B.author
- Clustered B+tree index on C.bid
- All index blocks are in memory
- Unlimited memory

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000

(a)

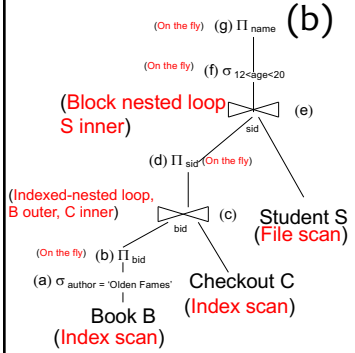


Cost =
T(B) / V(B, author)
= 50,000/500
= 100 (unclustered)

Cardinality =
100

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000

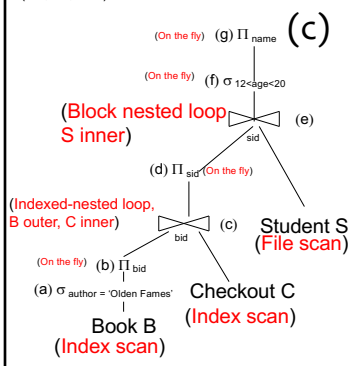
(b)



Cost =
0 (on the fly)

Cardinality =
100

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000



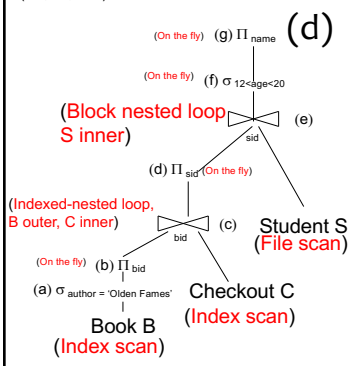
- one index lookup per outer B tuple
- 1 book has T(C)/ T(B) = 6 checkouts (uniformity)
- # C tuples per page = T(C)/B(C) = 20
- 6 tuples fit in at most 2 consecutive blocks (clustered) could assume 1 page as well

Cost <=
100 * 2= 200

Cardinality =
100 * 6 = 600

= 100 * T(C)/ MAX(100, V(C, bid)) assuming
V(C, bid) = V(B, bid) = T(B) = 50,000

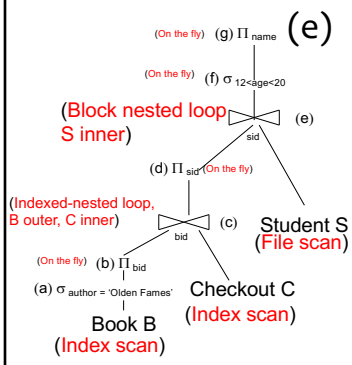
S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000



Cost =
0 (on the fly)

Cardinality =
600

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000

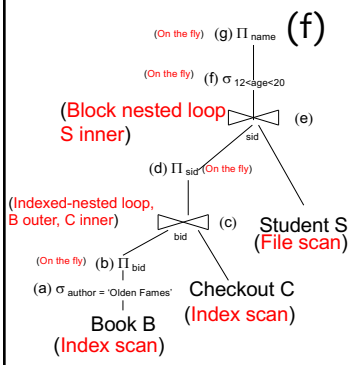


Outer relation is already in (unlimited) memory
need to scan S relation

Cost =
B(S) = 1000

Cardinality =
600

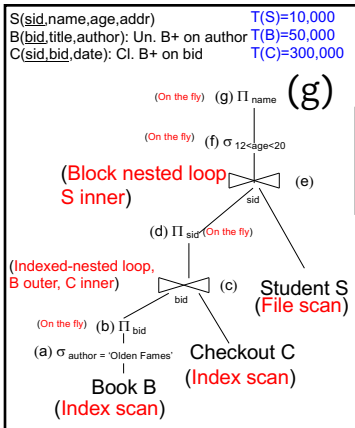
S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000



Cost =
0 (on the fly)

Cardinality =
600 * 7/18 = 234 (approx)

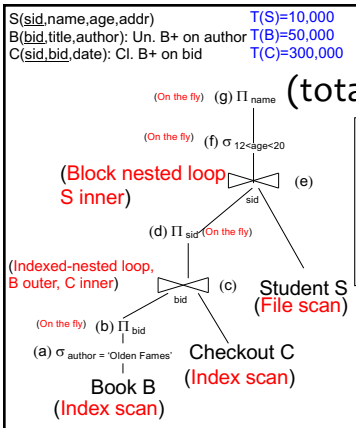
S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000



Cost =
0 (on the fly)

Cardinality =
234

S(sid,name,age,addr) T(S)=10,000 B(S)=1,000 V(B,author) = 500
B(bid,title,author): Un. B+ on author T(B)=50,000 B(B)=5,000 7 <= age <= 24
C(sid,bid,date): Cl. B+ on bid T(C)=300,000 B(C)=15,000



Total cost =
1300
(compare with 1,515,001,000 for plan 1!)

Final cardinality =
234 (approx)
(same as plan 1!)

43

Summary

- Relational algebra equivalence
- SQL rewrite tricks
- Heuristics-based optimization
- Cost-based optimization
  - Need statistics to estimate sizes of intermediate results
  - Greedy approach
  - Dynamic programming approach
- Combination of cost and size estimation along a plan