Implementing SELECT

Some possibilities for simple selection (a single condition):

name	strategy	restrictions
SL	brute force linear search (scan entire file)	always applicable
SB	binary search on the file	must have file ordered on attribute
SH	use hash key	must have file hashed on attribute must be equality condition
SP	use primary index	must have primary index on attribute
SC	use clustering index	must have clustering index on attribute
SS	use secondary index	must have secondary index on attribute

- only SL can be pipelined
  - SL can be carried out incrementally
  - everything else requires random access to the file or an index

Implementing JOIN R ⋈ S

Some possibilities for join:

name	strategy	restrictions
JNL	*nested-loop join* (brute force) for each record in R, go through every record in S and test if the join condition is satisfied	always applicable
JSL	*single loop join* for each record in R, use index to retrieve all matching records of S	requires index on join attribute for one file (S)
JSM	*sort-merge join* if necessary, sort files by join attribute scan files in order, matching records according to join attribute	always applicable
	scan indexes, matching records according to join attribute	requires indexes on join attribute for both files

- reading of R can be pipelined for JNL, JSL
- JSM can be pipelined for R and S if the records come out of the previous step in order (so no need to sort)

Implementing PROJECT

Without duplicate elimination, simply extract the desired columns for each record.

- can be pipelined

For duplicate elimination:

name	strategy
PS	sort file (duplicates will be adjacent)
PH	hash file (check each record against others in the bucket it hashes to)

- PS can be pipelined if the records come out of the previous step in order

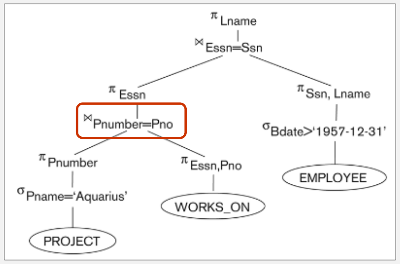
All questions make use of the following relational schema and query tree. Assume that the file for each table is ordered by the primary key, and that there are primary indexes for each file as well as indexes on PROJECT.Pname, WORKS_ON.Pno, and EMPLOYEE.Lname.

```
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Address,Sex,Salary,Super_ssn,Dno)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
```

JNL is always applicable

JSM is always applicable, though may require sorting
– PROJECT is ordered by Pnumber (no sorting)
– WORKS_ON is ordered by Essn,Pno (need sorting, or retrieve rows via WORKS_ON.Pno index)

JSL requires an index on S
– there is an index on WORKS_ON.Pno and π (without duplicate removal) can be pipelined

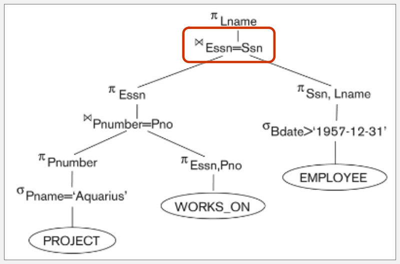


What algorithms are suitable for the $\bowtie_{Pnumber=Pno}$ operation? Choose all that apply.

JNL	4 respondents	80 %	✓
JSL	4 respondents	80 %	✓
JSM	4 respondents	80 %	✓
none of the above		0 %	

All questions make use of the following relational schema and query tree. Assume that the file for each table is ordered by the primary key, and that there are primary indexes for each file as well as indexes on PROJECT.Pname, WORKS_ON.Pno, and EMPLOYEE.Lname.

```
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Address,Sex,Salary,Super_ssn,Dno)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
```



JNL is always applicable

JSM is always applicable, though may require sorting – PROJECT is ordered by Pnumber so rows will likely come out of $\bowtie_{Pnumber=Pno}$ ordered by Pnumber (sorting needed) – EMPLOYEE is ordered by Ssn (need sorting)

JSL requires an index on S – there is an index on EMPLOYEE.Ssn but if that is used for the join, SL is required for $\sigma_{Bdate>'1957-12-31'}$ and both it and $\pi_{Ssn,Lname}$ must be pipelined (without duplicate removal)

What algorithms are suitable for the $\bowtie_{Essn=Ssn}$ operation? Choose all that apply.

JNL ✓	4 respondents	80 %
JSL	4 respondents	80 %
JSM ✓	3 respondents	60 %
none of the above		0 %

Query Optimization

Two main strategies:

- *heuristic* – apply rules-of-thumb for how to order operations
  - generally works well, but is not guaranteed to produce the best results
- *cost-based evaluation* – estimate the cost (e.g. number of disk blocks accessed) of different strategies, then choose best
  - need fast-to-compute cost functions
  - must limit number of strategies considered

Cost-Based Query Optimization

Many factors contribute to the cost of evaluating a query:

- access to secondary storage
  - reading/writing disk blocks

  dominates in large databases (not everything fits into memory at once, and disks are slow)
- storage cost
  - storing intermediate results, including reading/writing disk blocks
- computation cost
  - in-memory searching, sorting, merging

  dominates in small databases (most data fits in memory)
- memory usage cost
  - memory used for query processing is not available for other uses
- communication cost
  - sending query and results between client and database

  dominates in distributed databases

Cost Functions

Assumptions:

- storage costs dominate (large DB)
- multilevel indexes

Cost is measured in terms of the number of blocks read and written.

Note.
Similar tactics can be employed to work out cost functions given other assumptions e.g. small DB where in-memory computation cost dominates (cost is measured in terms of the number of records accessed) or only single-level indexes.

Example Cost Functions – SELECT

algorithm	condition	# blocks read
SL – linear search (brute force)	equality, key	b/2 ← on average
	equality, not key range	b
SB – binary search	equality, key	$\log_2 b$
	equality, not key	$\log_2 b + \lceil s/bfr \rceil$
	range	$\log_2 b + b/2$ ←
SH – hash file	equality, key	1 or 2
SP – primary index	equality, key	x + 1
	range	x + b/2 ←
SC – clustering index	equality	$x + \lceil s/bfr \rceil$
	range	x + b/2 ←
SS – secondary index	equality, key	x + 1
	equality, not key	x + s
	range	$x + b_1/2 + r/2$ ←

for range searches, assume roughly half the file records (and thus half the blocks if the file is ordered on the field involved in the condition) will satisfy the condition – can be very inaccurate in specific cases, but reasonably correct on average (can use better estimate if relevant DB stats are available)

Number of blocks in result: s/bfr

x = # levels in index, s = selection cardinality (# matches)

Attribute Selectivity and Selection Cardinality

- attribute selectivity (*sl*)
  - fraction of records satisfying equality condition on the attribute
- selection cardinality (*s*) = *sl***r*
  - average number of records satisfying equality condition
  - for key, $d = r$, $sl = 1/r$, $s = 1$
  - for uniformly distributed non-key, $sl = 1/d$, $s = r/d$

Example Cost Functions – JOIN $(R \bowtie_{R.A=S.B} S)$

name	implementation	# blocks read
JNL – nested-loop join	read in n-2 blocks of R at a time read all blocks of S for each n-2 blocks of R, one at a time use one block of memory for assembling result	$b_R + \lceil b_R/(n-2) \rceil b_S$
JSL – single-loop join	read in a block of R, find all matches of S using the index	secondary index: $b_R + \|R\| (x_B+s_B)$
		clustering index: $b_R + \|R\| (x_B+\lceil s_B/bfr_S \rceil)$
		primary index: $b_R + \|R\| (x_B+1)$
		hash key: $b_R + \|R\|$
JSM – sort-merge join	sort files (blocks read and written)	external sorting: $2b + 2b \log_{\min(n-1,b/n)} (b/n) \approx 2b \log_2 b$
	merge	$b_R + b_S$

Number of blocks in result: $(js \|R\| \|S\| / bfr_{RS})$

x_B = # levels in index on field B, js = join selectivity, n = # memory blocks to use

Join Selectivity

Join selectivity (js) is the fraction of possible records that actually result from a join.

$$js = | R \bowtie S | / |R| |S|$$

$0 \leq js \leq 1$

Some noteworthy special cases:

- for $R \times S$, js = 1
- for the join condition R.A=S.B
  - if A is a key of R, $js \leq 1/|R|$
  - if B is a key of S, $js \leq 1/|S|$
  - if both are keys, $js = \min(1/|R|,1/|S|)$
  - if neither is a key, $js = \min(s_B/|S|,s_A/|R|)$

External Sorting

Sorting is an important algorithm in query processing.

- ORDER BY requires sorting
- used in duplicate elimination (SELECT DISTINCT)
- can be used in carrying out JOIN (JSM), UNION

What's the best sorting algorithm?

External Sorting

Sorting in the context of a DB has some particular issues.

- lots of data → speed is important
- lots of data → which likely doesn't all fit into memory at one time
  - reducing the number of disk blocks accessed is more important than reducing the number of in-memory operations

External sorting refers to sorting where the data does not fit entirely into memory.

External Sorting

Typical approach is *sort-merge*, similar to merge sort.

- sorting phase sorts one chunk of the file at a time (as much as will fit into memory) using a traditional internal sort
  - number of runs $n_R = b/n$ (each with n blocks)
- merging phase merges sorted runs in a series of passes
  - after first pass, the n_R runs have been merged into n_R/d_M new runs
    - d_M = number of runs that can be merged in one pass = $\min(n-1, n_R)$
  - after second pass, the n_R/d_M runs have been merged into $n_R/(d_M)^2$ new runs
  - …

Overall runtime (number of blocks read/written) is $2b + 2b \log_{d_M}(n_R)$, which can be approximated by $\mathbf{2b \log_2 b}$.

Evaluating Query Cost

What information do we need for evaluating cost functions?

- number of records (r), number of blocks (b)
- blocking factor (bfr)
  - can be calculated directly or estimated from b, r
  - bfr_{RS} can be calculated explicitly if column sizes are known, or estimated as $\lceil 1/(b_R/r_R + b_S/r_S) \rceil = \lceil bfr_R bfr_S/(bfr_R + bfr_S) \rceil$
- physical file organization, available indexes, number of levels (x) of multilevel indexes, number of first-level index blocks (b_1)
- number of distinct values (d)
- attribute selectivity (sl), selection cardinality (s) for each attribute
  - can be computed from d and r (requires assumption of uniform distribution or knowledge of distribution if non-key)

Obtaining the Necessary Information

DBMS stores this information.

- frequently-changed values may not be kept completely up-to-date

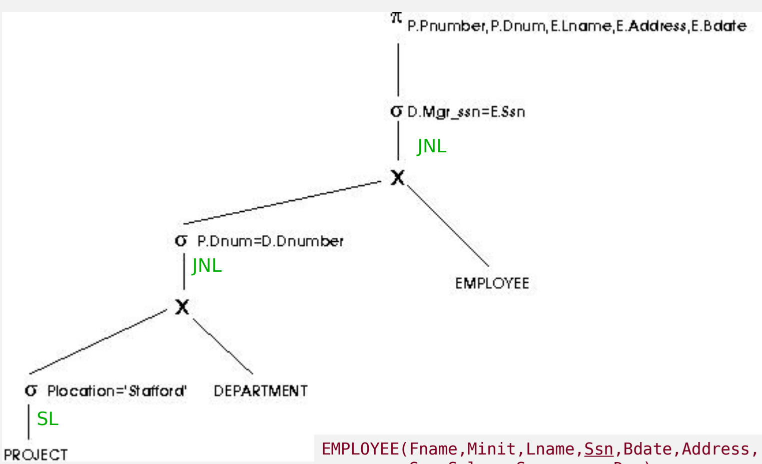
DB Stats

table	column	# distinct values (d)	low value	high value
PROJECT	Plocation	200	1	200
	Pnumber	2000	1	2000
	Dnum	50	1	50
DEPARTMENT	Dnumber	50	1	50
	Mgr_ssn	50	1	50
EMPLOYEE	Ssn	10000	1	10000
	Dno	50	1	50
	Salary	500	1	500
	Bdate		1945-01-01	1989-12-31

table	# records (r)	# blocks (b)
PROJECT	2000	100
DEPARTMENT	50	5
EMPLOYEE	10000	2000

index	# levels (x)	# level 1 blocks (b_1)
PROJ_PLOC	2	4
EMP_SSN	2	50
EMP_SAL	2	50

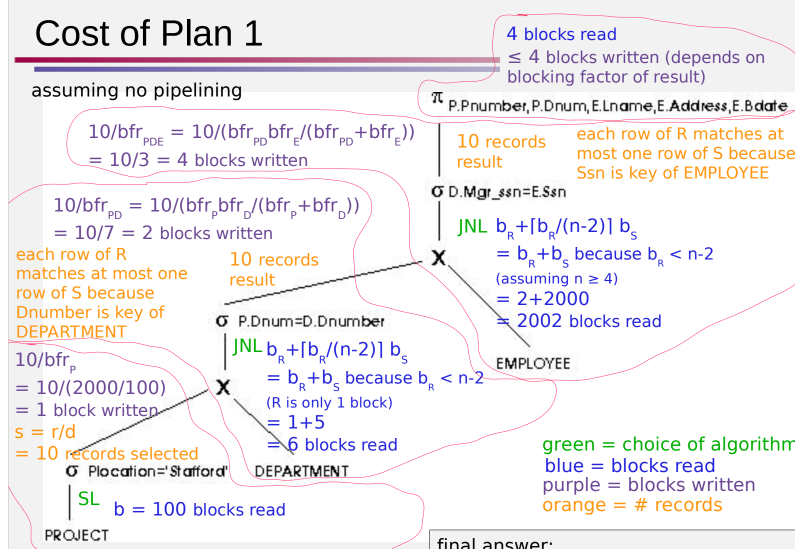
Cost of an Execution Plan – Plan 1




```
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Address,
         Sex,Salary,Super_ssn,Dno)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
```

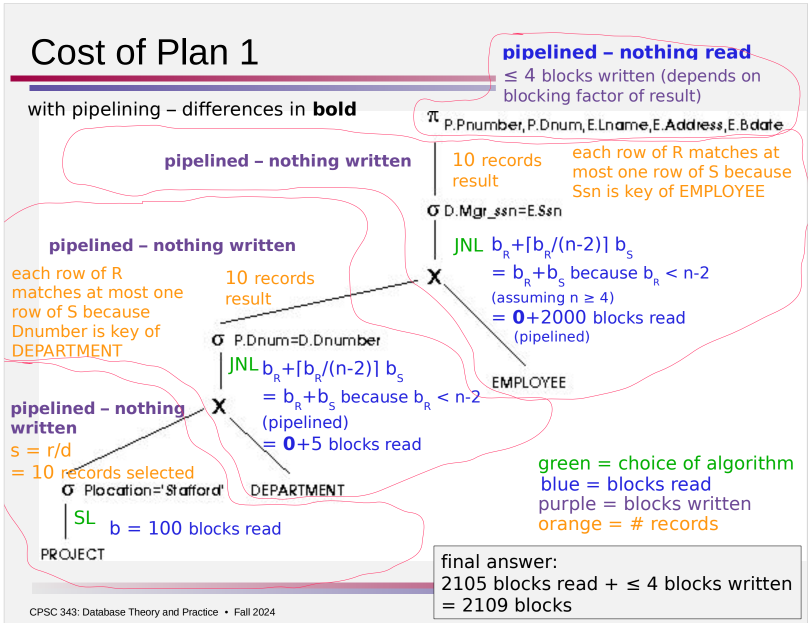

Cost of Plan 1

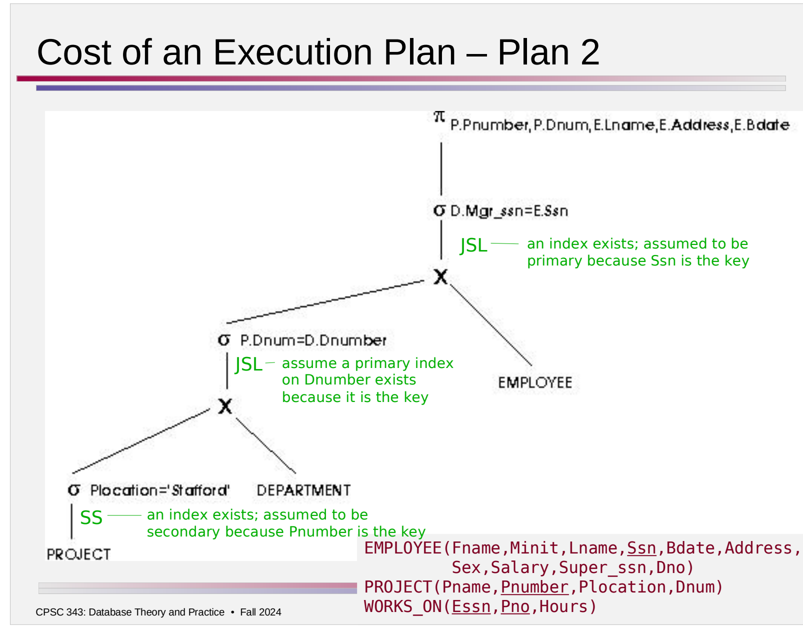
assuming no pipelining

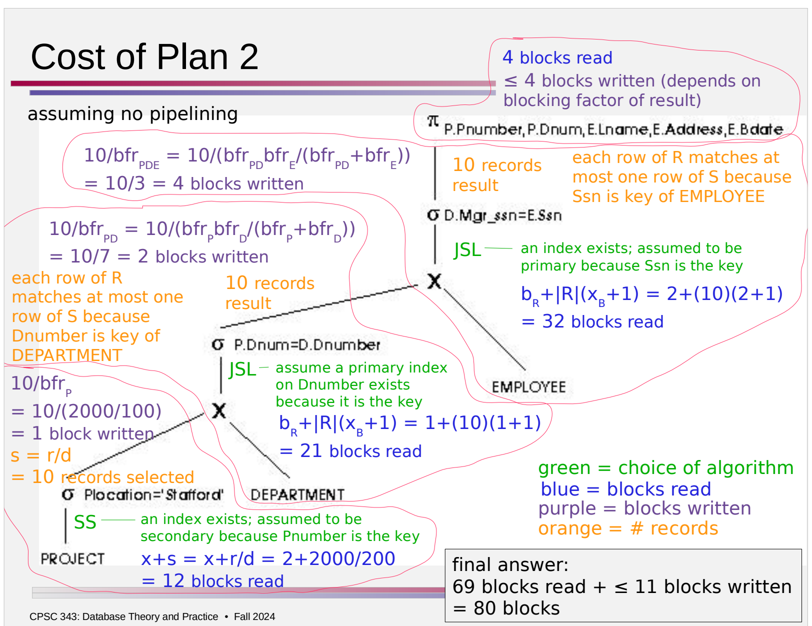


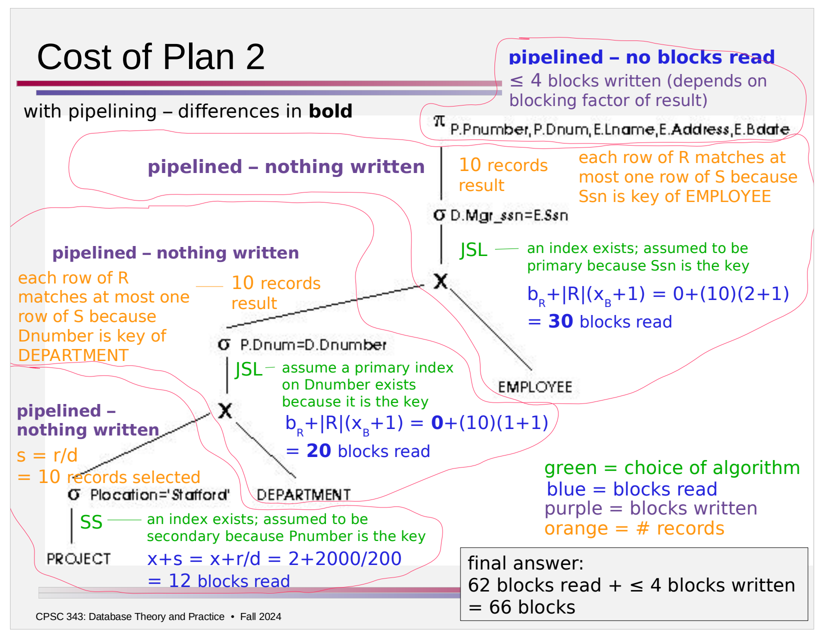
green = choice of algorithm
blue = blocks read
purple = blocks written
orange = # records

final answer:
2112 blocks read + ≤ 11 blocks written
= 2123 blocks


Cost of Plan 1
with pipelining – differences in **bold**
pipelined – nothing read
≤ 4 blocks written (depends on blocking factor of result)
π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
pipelined – nothing written
10 records result
each row of R matches at most one row of S because Ssn is key of EMPLOYEE
σ D.Mgr_ssn=E.Ssn
JNL $b_R+\lceil b_R/(n-2)\rceil\ b_S$
$= b_R+b_S$ because $b_R < n-2$
(assuming n ≥ 4)
= **0**+2000 blocks read
(pipelined)
pipelined – nothing written
each row of R matches at most one row of S because Dnumber is key of DEPARTMENT
10 records result
X
σ P.Dnum=D.Dnumber
JNL $b_R+\lceil b_R/(n-2)\rceil\ b_S$
$= b_R+b_S$ because $b_R < n-2$
(pipelined)
= **0**+5 blocks read
pipelined – nothing written
s = r/d
= 10 records selected
X
EMPLOYEE
σ Plocation='Stafford'
DEPARTMENT
SL
b = 100 blocks read
PROJECT
green = choice of algorithm
blue = blocks read
purple = blocks written
orange = # records
final answer:
2105 blocks read + ≤ 4 blocks written
= 2109 blocks
CPSC 343: Database Theory and Practice • Fall 2024


Cost of an Execution Plan – Plan 2
π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
σ D.Mgr_ssn=E.Ssn
JSL an index exists; assumed to be primary because Ssn is the key
X
σ P.Dnum=D.Dnumber
JSL assume a primary index on Dnumber exists because it is the key
EMPLOYEE
X
σ Plocation='Stafford'
DEPARTMENT
SS an index exists; assumed to be secondary because Pnumber is the key
PROJECT
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Address,
Sex,Salary,Super_ssn,Dno)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
CPSC 343: Database Theory and Practice • Fall 2024


Cost of Plan 2
assuming no pipelining
4 blocks read
≤ 4 blocks written (depends on blocking factor of result)
π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
$10/bfr_{PDE} = 10/(bfr_{PD}bfr_E/(bfr_{PD}+bfr_E))$
= 10/3 = 4 blocks written
10 records result
each row of R matches at most one row of S because Ssn is key of EMPLOYEE
σ D.Mgr_ssn=E.Ssn
$10/bfr_{PD} = 10/(bfr_P bfr_D/(bfr_P+bfr_D))$
= 10/7 = 2 blocks written
JSL an index exists; assumed to be primary because Ssn is the key
each row of R matches at most one row of S because Dnumber is key of DEPARTMENT
10 records result
X
$b_R+|R|(x_B+1) = 2+(10)(2+1)$
= 32 blocks read
σ P.Dnum=D.Dnumber
$10/bfr_P$
= 10/(2000/100)
= 1 block written
s = r/d
= 10 records selected
JSL assume a primary index on Dnumber exists because it is the key
EMPLOYEE
X
$b_R+|R|(x_B+1) = 1+(10)(1+1)$
= 21 blocks read
σ Plocation='Stafford'
DEPARTMENT
SS an index exists; assumed to be secondary because Pnumber is the key
PROJECT
x+s = x+r/d = 2+2000/200
= 12 blocks read
green = choice of algorithm
blue = blocks read
purple = blocks written
orange = # records
final answer:
69 blocks read + ≤ 11 blocks written
= 80 blocks
CPSC 343: Database Theory and Practice • Fall 2024


Cost of Plan 2
with pipelining – differences in **bold**
pipelined – no blocks read
≤ 4 blocks written (depends on blocking factor of result)
π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
pipelined – nothing written
10 records result
each row of R matches at most one row of S because Ssn is key of EMPLOYEE
σ D.Mgr_ssn=E.Ssn
JSL an index exists; assumed to be primary because Ssn is the key
pipelined – nothing written
each row of R matches at most one row of S because Dnumber is key of DEPARTMENT
10 records result
X
$b_R+|R|(x_B+1)$ = 0+(10)(2+1)
= **30** blocks read
σ P.Dnum=D.Dnumber
JSL assume a primary index on Dnumber exists because it is the key
EMPLOYEE
pipelined – nothing written
s = r/d
= 10 records selected
X
$b_R+|R|(x_B+1)$ = **0**+(10)(1+1)
= **20** blocks read
σ Plocation='Stafford'
DEPARTMENT
SS an index exists; assumed to be secondary because Pnumber is the key
PROJECT
x+s = x+r/d = 2+2000/200
= 12 blocks read
green = choice of algorithm
blue = blocks read
purple = blocks written
orange = # records
final answer:
62 blocks read + ≤ 4 blocks written
= 66 blocks
CPSC 343: Database Theory and Practice • Fall 2024