CSE 414 Database Systems

Section 5: Midterm Review,

Cost estimation

TA: Daseul Lee (dslee@cs)

Midterm

- Midterm: on Monday, May 6th
- Open book + 1 sheet of paper with handwritten notes
- Review session
  - Time: 2PM on Sunday, May 5th
  - Location: LOEW 101
  - Q&A session: bring questions with you

Midterm Topics

1. SQL
2. Relational Algebra
3. Query Implementation

1. SQL - Terminology

- Database is a collection of relations
- Relation / Table
  - Relation schema: structure of a relation
  - Instance: actual data content
    - Row / Tuple / Record
    - Column / Attribute / Field

1. SQL - Key

- Within a relation:
  - Key (candidate key): one or more attributes that uniquely identifies a record
  - Primary key: one candidate key selected for a relation
  - Foreign key: one or more attributes that reference a candidate key from another relation
    - logical pointer to a parent table
- Sequential file: how data file is sorted, if at all
- Index file: how index is organized

1. SQL - Syntax

- Basic commands:
  - CREATE – creates a new table

    ex) CREATE TABLE [table] (…);
  - INSERT INTO - inserts new data into a table

    ex) INSERT INTO [table] VALUES ([value1], [value2], …);
  - SELECT - extracts data from a table

    ex) SELECT [column(s)] FROM [table_name];
  - UPDATE - updates data in a table

    ex) UPDATE FROM [table] SET … WHERE …;
  - DELETE - deletes data from a table

    ex) DELETE FROM [table] WHERE …;

1. SQL - Syntax

- Other clauses:
  - WHERE
  - GROUP BY
  - HAVING
  - ORDER BY … [DESC]
- Operators:
  - DISTINCT, AS, LIKE, AND, OR, =, >, < ….
  - EXISTS, NOT EXISTS, IN, NOT IN, ALL, ANY
  - JOIN, LEFT OUTER JOIN … ON … , etc

1. SQL - Join

- (Inner) Join vs. Outer Join
  - Outer Join includes the tuples with no match, filled with NULL
  - Outer Join can be
    - Left Outer Join
    - Right Outer Join
    - Full Outer Join
- Important to carefully select the type of join in order to query all the data of your interest

1. SQL – Aggregates

```
SELECT   S
FROM     R_1,…,R_n
WHERE    C1
GROUP BY a_1,…,a_k
HAVING   C2
```

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on aggregate expressions and on attributes $a_1,\ldots,a_k$

1. SQL – Aggregates

- Aggregate functions:

  count(), sum(), avg(), min(), max()

  -> applied to a single attribute, except count which can count the number of rows, i.e. count(*)
- Make sure to determine whether you want to apply the function on duplicate or distinct values.
- If used with GROUP BY, then aggregate function is applied to "each" group.
- Otherwise, the function is performed on the whole output relation.

1. SQL – Nested query

- In SELECT (less common): returns a constant or computed value
- In FROM (less common): returns a relation, followed by a variable -> useful for "finding witnesses"
- In WHERE (common): returns a constant or a relation
  - Existential quantifier: EXISTS, IN, ANY
    -> easy to un-nest
  - Universal quantifier: ALL, NOT EXISTS + [negated condition in subquery], NOT IN + [negated condition in subquery]
    -> hard to un-nest

1. SQL – Nested query

- "Finding witnesses": Let's say we want to query a tuple that contains the max value on some attribute x, not the max itself
  - Solution 1) have two instances of a table; one for finding a max and the other for comparing x in each tuple with the max
  - Solution 2) use subquery to return the max value and, in outer query compare x in each tuple with the max
  - Solution 3) use subquery to return x in each tuple and, in outer query find a tuple s.t. x >= ALL { select x ... }

1. SQL – Indexing

- An additional file, that allows fast access to records in the data file given a search key
- Classification
  - Clustered / unclustered
  - Primary / secondary
  - (Organization) Hash table / B+ tree
- Trade-offs: faster query vs. slower update
- Index selection problem
  - Consider workload
  - Attributes that appear in WHERE
  - Covering index? (multiple attributes)
  - Clustered or not?

2. RA – Relational operators

- Union $\cup$, intersection $\cap$, difference -
- Selection σ
- Projection Π
- Cartesian product $\times$, join $\bowtie$
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

RA
(Set semantics)

Extended RA
(Bag semantics)

2. Relational Algebra – More on Joins

- **Theta-join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_\theta S = \pi_A(\sigma_\theta(R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_A drops all redundant attributes
- **Natural join**: $R \bowtie S = \pi_A(\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** fields with same name in R and in S

2. Relational Algebra – SQL <-> RA

- Given a SQL query, translate it into an equivalent relational algebra expression or logical query plan (tree) – or, vice versa
- Can you come up with a more efficient plan?
  - In general, joins are expensive; pushing down the selection before the join can reduce the size of input tuples, if any.
  - And many more optimizations can be done …
- Nested queries? Try removing a correlation between the outer query and the inner query

3. Query Implementation

- Logical query plan: an extended relational algebra tree
  - Logical query plan may have several physical query plans
- Physical query plan: a logical query plan with extra annotation
  1. Access path selection for each relation
  2. Implementation choice for each operator
  3. Scheduling decision (not required)

3. Query Implementation – Physical query plan

1. Access methods: Heap file, Hash-based index, Tree-based index
2. Operator implementations, ex) Join
   – Nested loop join
   – Hash join
   – Sort-merge join
   – Index nested loop join

Hash join, Sort-merge join: One-pass or Two-pass algorithm

Index nested loop join: Index-based algorithm

One-pass if operator reads its operand only once and no need to write intermediate results into the disk

3. Query Implementation – Cost estimation

- Cost: total number of I/O operations
  - I/Os are performed at page (or block) level
- Parameters:
  - B(R) = # of blocks for relation R
  - T(R) = # of tuples in relation R
  - V(R, a) = # of distinct values of attribute a
- Main constraint:
  - M = # of memory (buffer) pages

3. Query Implementation – Cost estimation on Join operation

- Nested Loop Join
  - $B(R) + T(R)\,B(S)$
  - $B(R) + B(R)B(S)$ with Page-at-a-time Refinement
- Hash Join
  - $B(R) + B(S)$ when $B(R) <= M$ (1st pass)
  - $3B(R) + 3B(S)$ when $\min(B(R), B(S)) <= M^2$ (2nd pass)
- Sort-Merge Join
  - $B(R) + B(S)$ when $B(S)+B(R) <= M$ (1st pass)
  - $5B(R)+5B(S)$ when $B(R) <= M^2$, $B(S) <= M^2$ (2nd pass)

    or, $3B(R)+3B(S)$ when $B(R) + B(S) <= M^2$ and compute the join during the merge phase
- Index Nested Loop Join
  - If index on S is clustered: $B(R)+T(R)B(S) / V(S,a)$
  - If index on S is unclustered: $B(R)+T(R)T(S)/V(S,a)$

General Tips

- Go over lecture notes
- Try example problems from the sections
- Try old 344 midterms (Do not worry about datalog and relational calculus)
- Bring questions to the review session
- Questions?

More on Cost Estimation

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

Example

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

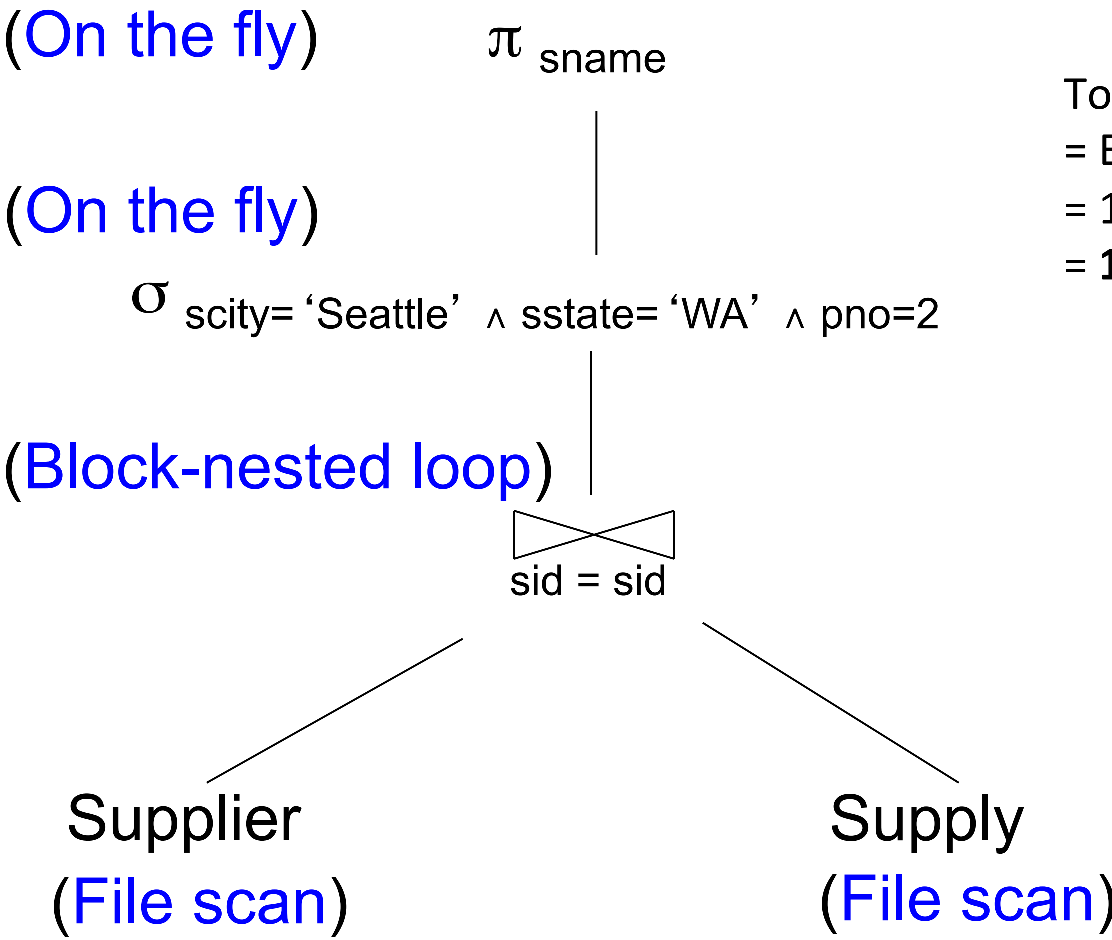
T(Supplier) = 1000 records
T(Supply) = 10,000 records
B(Supplier) = 100 pages
B(Supply) = 100 pages
V(Supplier, scity) = 20
V(Suppliers, state) = 10
V(Supply ,pno) = 2,500
M = 11 pages
* Both relations are clustered

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

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

T(Supply) = 10,000
B(Supply) = 100
V(Suppliers,state) = 10
V(Supply,pno) = 2,500

Physical Query Plan 1



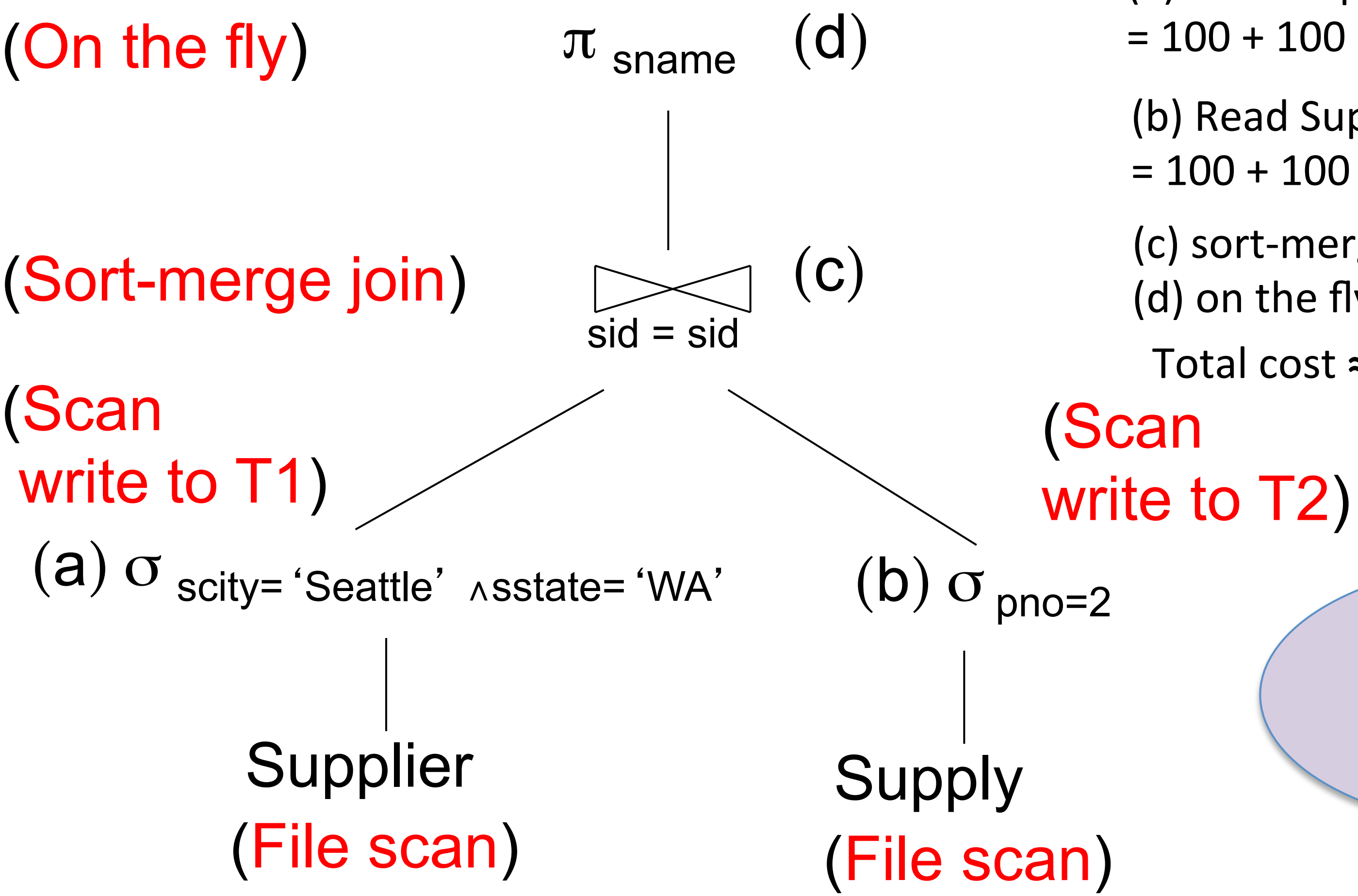
Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)
= 100 + 100 * 100
= **10,100 I/Os**

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

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

T(Supply) = 10,000
B(Supply) = 100
V(Suppliers,state) = 10
V(Supply,pno) = 2,500

Physical Query Plan 2



(a) Read Supplier + Write T1
= 100 + 100 * 1/20 * 1/10 ≈ 101

(b) Read Supply + Write T2
= 100 + 100 * 1/2500 ≈ 101

(c) sort-merge join on T1 & T2= 2
(d) on the fly = 0

Total cost ≈ **204 I/Os**

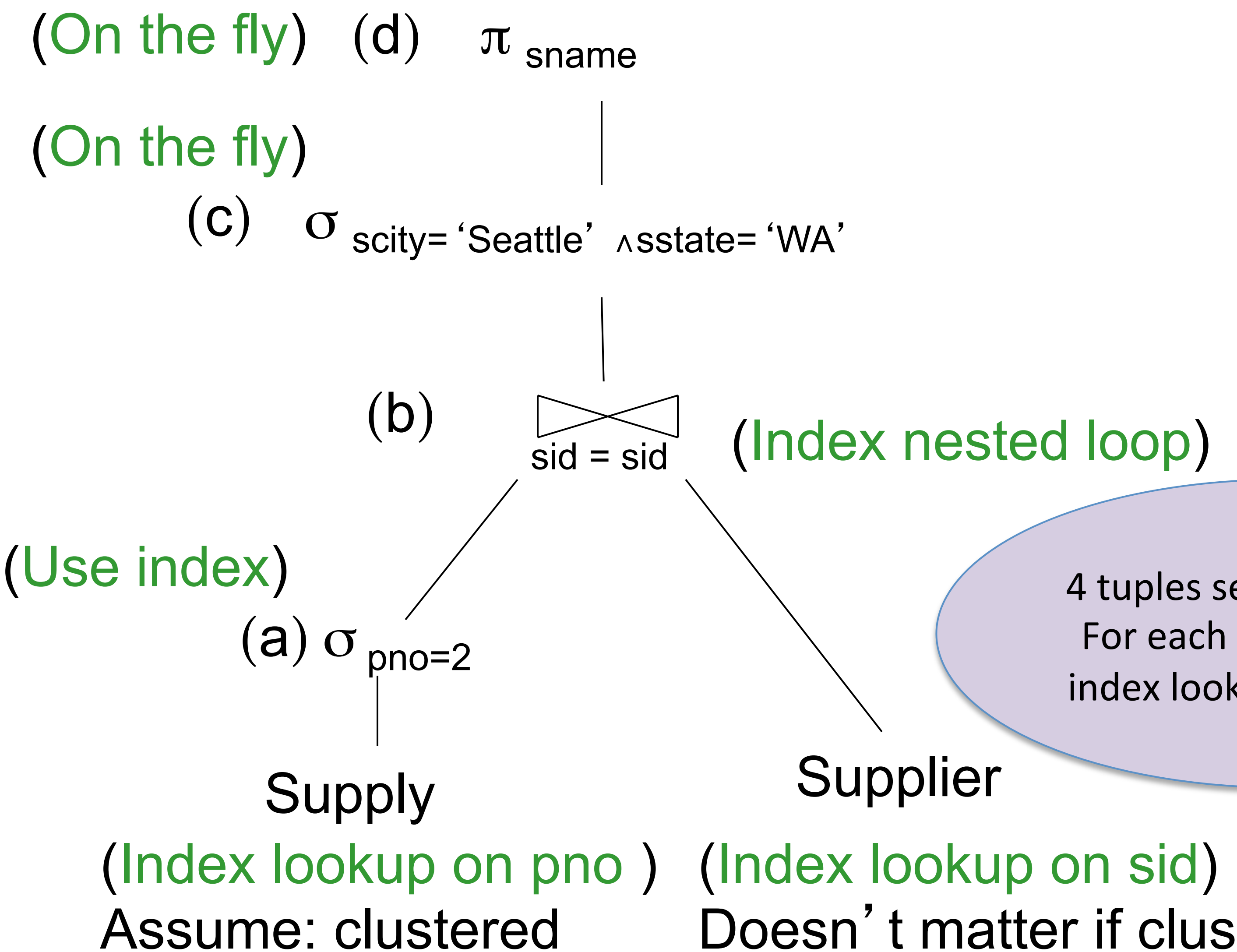
T1 and T2 has at most one page each

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

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

T(Supply) = 10,000
B(Supply) = 100
V(Suppliers,state) = 10
V(Supply,pno) = 2,500

Physical Query Plan 3



(a) 100 / 2500 ≈ 1
(b) 4 * 1 = 4
(c) 0
(d) 0
Total cost ≈ **5 I/Os**

4 tuples selected from (a)
For each tuple, perform
index look up on Supplier