Thursday 23/4/2015 Query Execution

Lecture Topics

I. Query Plan
II. Finding Records
III. Operator Re-ordering
IV. Algorithm Selection

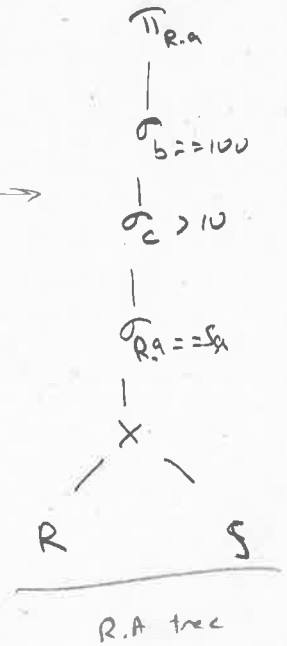
I. Query Plan

A query plan is an extended R.A. tree with annotations for the access / implementation method.

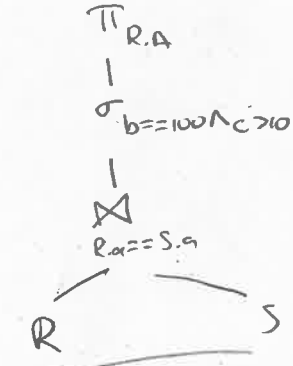
Ex. R(a,b) and S(a,c)

```
Select R.a
FROM R, S
WHERE R.a == S.a
  AND R.b == 100
  AND S.c > 10
```

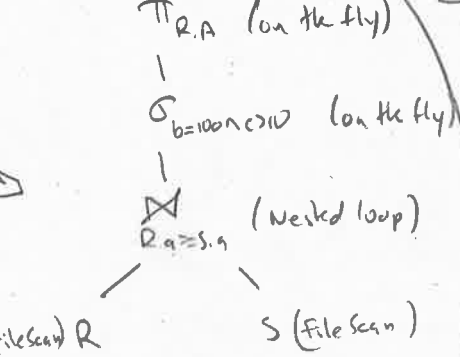
Query



R.A tree



Equivalent R.A. tree



Query Plan

- For any given query, there may be various ways to execute it.
- The DBMS must:
  - consider different plans (i.e. need an algorithm to search plan space)
  - maintain statistics about the database and workload (e.g. profiling)
  - Estimate the cost of a plan
  - decide which plan to use.
    - Ideally choose the best
    - usually don't choose the worst.

II Finding Records

Example:

Say we have a file with 100,000 records, 100 records per block, (1,000 blocks)

- There is a hash index on the records
- File is unclustered
- To answer a query, we need to read all blocks that contain relevant records.

Scenario 1: Find 50 records.
Probably need to access 50 (or almost) 50 blocks. Good to use an index.

Scenario 2: Find 2,000 records.
Note only 2% of all records, but, very likely that the records are in 1,000 blocks (or almost). Better to traverse the file rather than use an index.

Example:

```
Select SSN
FROM Employee
WHERE DOB == "1982-05-03"
  AND SEX == 'F'.
```

Two strategies:

(1) Find all blocks for DOB == "1982-05-03" and then pick the records where SEX == 'F';

(2) Find all blocks for SEX == 'F' and then pick records where DOB == "1982-05-03"

If 50% of employees are F, then (2) is a bad strategy.

Either system must profile, or SQL programmer must influence plan.

III Operator Re-ordering

```
SELECT *
FROM R
WHERE A=1
  AND B='MARY'
```

$\Rightarrow$ $\sigma_{A=1}(\sigma_{B='MARY'}(R))$ OR $\sigma_{B='MARY'}(\sigma_{A=1}(R))$

IMAGINE THERE ARE INDICES ON A AND B.

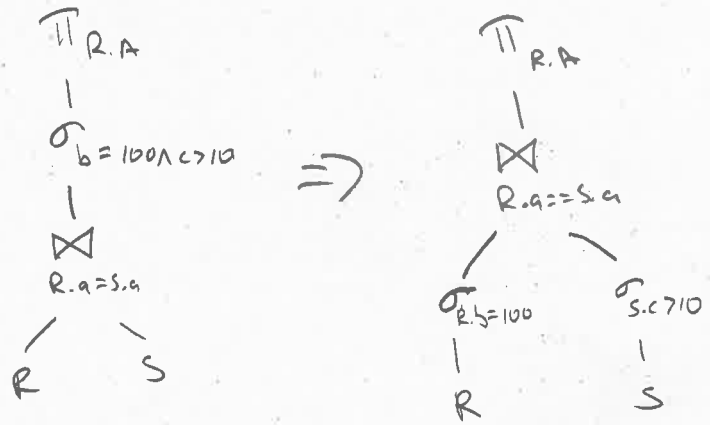
$\Rightarrow$ CAN EASILY FIND ONE RECORD WITH A=1 OR ONE RECORD WITH B='MARY'

MORE EFFICIENT TO DO THE more selective operation first

- some databases profile, decide which index is most selective.
- some pick first in sequence (e.g. A)
- some pick last in sequence (e.g. B.)
  - Oracle used to do this.
- Programmer may have to re-write their query:

```
WHERE A=1 AND B='MARY'
OR
WHERE B='MARY' AND A='1'
```

- Pushing Selection:
- A join is an expensive operation.
  - $\Rightarrow$ simple strategy is nested loops. $O(n^2)$



23/4/2015 Query Execution Pg3

Order of Joins:

Example:

Lives (Person, City) about people in the U.S.
~ 300,000,000 tuples

Oscar (Person) people in the U.S. who have won an Oscar ~ 1000 tuples

Nobel (Person) people in the U.S. who have won a Nobel ~ 100 tuples

Query: Find Good_Match (Person1, Person2) where the two people live in the same city, and the first person won an oscar and the second won a nobel

- How would you do this "by hand"?

- With SQL:

```
SELECT Oscar.Person as P1, Nobel.Person as P2
FROM Oscal, Lives Lives1, Nobel, Lives Lives2
WHERE Oscal.person == Lives1.Person
AND NOBEL.person = Lives2.person
AND LIVES1.CITY = LIVES2.city
```

VERY SLOW!!!

Alternative:

create oscar-pc (person, city) people with oscars and their city

create nobel-pc (person, city) people with nobel and their city

join on the smaller tables.

effectively doing the selection earlier

IV Algorithm Selection

Example:

R(A, B)
S(C, D)

- No indexes
- 1,000 blocks in R
- 10,000 blocks in S

Available 3 blocks of RAM

Query:

```
SELECT *
FROM R, S
WHERE R.B = S.C
```

STRATEGY 1:

READ 2 BLOCKS OF R INTO MEMORY
- READ 1 BLOCK OF S INTO MEMORY
- CHECK FOR JOIN CONDITION ON ALL RESIDENT BLOCKS.
- REPEAT 10,000 x (FOR ALL S BLOCKS)

REPEAT FOR ALL SUBSETS OF 2 OF R (TOTAL 500 x)

TOTAL READS =

1 READ OF R + 500 READS OF S =

5,001,000 blocks READ

STRATEGY 2:

READ 2 Blocks of S INTO Memory
- Read 1 block of R
- check join condition
- REPEAT 1,000 x (FOR ALL R Blocks)

REPEAT FOR ALL S subsets (TOTAL 5,000 Times)

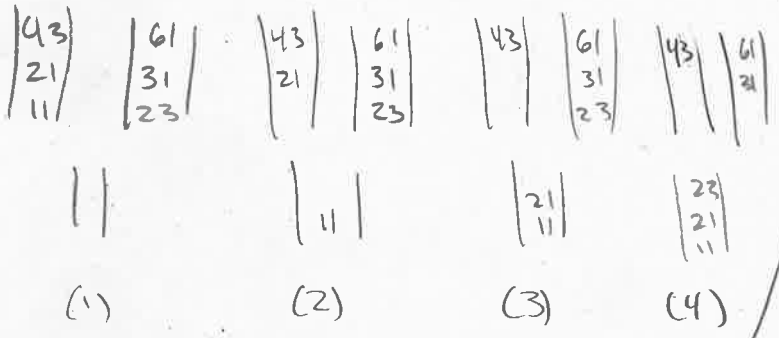
TOTAL READS =

1 READ of S + 5,000 READS OF R =

5,010,000 blocks read

Merge sort (Reminder):

- Divide & conquer strategy
- At every iteration, we have sorted sequences of some length
- After each stage, sequences double in length, but there are half
- We can do this with 3 blocks
  - two to hold the sorted sequences
  - one to hold the merged output



Merge Join

- If R is sorted on B and S is sorted on C, we can use merge sort to join them

Merge:

- Compare current candidates, output smaller

JOIN:

- Compare current candidates, join if equal, else discard smaller.

Algorithm:

- Sort R (merge sort)
- Sort S (merge sort)
- Join R and S

To sort R:

- Read 3 blocks of R, sort in place
- Merge sort R

To sort S: same

Performance

To sort R, about 10 passes (to read and write) (i.e. $\log_2 1{,}000$)

To sort S, about 14 passes (i.e. $\log_2 10{,}000$)

To merge-join R and S: one pass 1,000 reads for R, 10,000 reads for S. At most, 1,000 writes for the join. (bounded by R)

Total cost less than 312,000 disk accesses.

Hash-Join

- sorting is similar to creating an index.
- could use a hash, instead of sorted order.

IDEA: Hash join attribute for both relations using the same hash function.

If we hash relations into k partitions, we know that R tuples in partition i can only join with S tuples in partition i.

Read partition i for smaller relation R.
Scan partition of S for matches.
Never need to consider these tuples again.