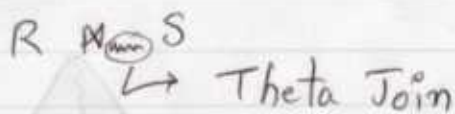


1-APR-09

Database Management Systems

Join with general Conditions



1. Equality Joins with multiple attributes:

```
SELECT
FROM R, S
WHERE R.a = S.b AND R.c = S.d
```

for $r \in S$, index will help.

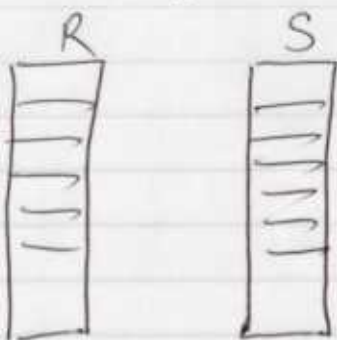
Index on $\langle b, d \rangle$, $\langle a, c \rangle$ will be the best scenario.

If not, we can build ^{such} an index ($N \log N$) and then search.

Non-EQUALITY JOINS

```
SELECT
FROM R, S
WHERE R.a < S.b
```

① Does sorting work? No!



Sorted tables

② Hash does not work - obviously!!

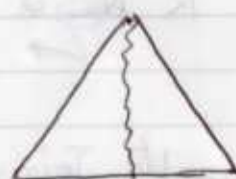
NON EQUALITY JOINS CONTINUED...

3) INDRA BASED

for $r \in S$

$$r.a < s.b$$

use as key in B^+ tree



→ matches.

for unclustered, seq I/O for each tuple → could be worse than w/o index

clustered - Good.

Block based NLJ is next choice - Not too bad.

END OF JOIN ALGORITHMS.

IV SET OPERATORS $\left(\frac{U}{n}\right)$

For cross product - No efficiency - Read all tuples.

i) Set Difference

$$R - S$$



Sorting/Hash based is the way.

III Aggregates:

Linear processing is needed.

```
To use index only plan,  
SELECT COUNT(*)  
FROM SAILORS,  
WHERE AGE > 30
```

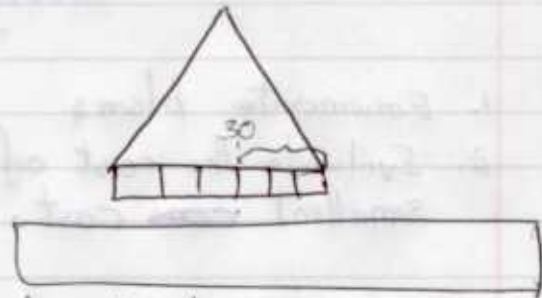


1/4/09

DBMS

AGGREGATES CONTO...

if we have index on age, no I/O needed.



If there exists an index on all attributes in the WHERE clause index only plans are possible.

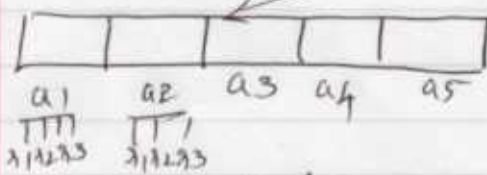
```

SELECT rating, count(*)
FROM sailors
WHERE AGE > 30
GROUP BY RATING.

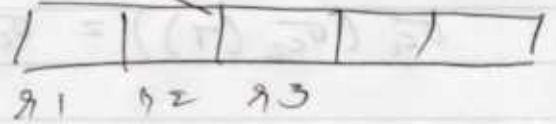
```

index on all attributes in WHERE & GROUP BY for index only plan.

<rating, age>
 <age, rating>



NOT Good
needs a scan



good! ↓ faster response time
just kick out where age > 30

END OF SECTION

QUERY OPTIMIZATION

1. Enumerate plans
2. Evaluate the cost of plans and choose one with the smallest ~~cost~~ cost.

The time taken to search should be optimized and is hard.

Plan Generation:



equivalence rules (17-18 major types).

Cascading Rule:

$$\sigma_{c_1} \sigma_{c_2} \sigma_{c_3} \dots (T) = \sigma_{c_1} (\dots \sigma_{c_{n-1}} (\sigma_{c_n} (T)) \dots)$$

Commutative:

$$\sigma_{c_1} (\sigma_{c_2} (T)) = \sigma_{c_2} (\sigma_{c_1} (T))$$

$$\pi_{a_1} (\pi_{a_2} (\pi_{a_3} \dots \pi_{a_n} (T)))$$

$$a_1 \subseteq a_2 \subseteq a_3 \dots \subseteq a_n$$

$$\pi_{a_1} (T)$$

A

Associativity of Joins:

$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

Commutative:

$$R \bowtie S = S \bowtie R$$

DBMS

$$\pi_a(\sigma_c(T)) = \sigma_c(\pi_a(T))$$

Helps in getting a smaller table earlier.
Required that the result be the same

$\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie S$
all attributes in 'c' are in 'R'
use the smaller table to do a join.

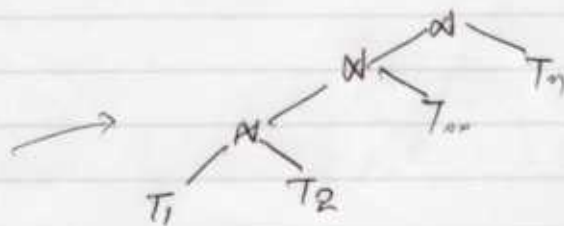
Eg: $\sigma_{R.a > 5 \wedge S.b > 30}(R \bowtie S)$
 $= \sigma_{R.a > 5}(R) \bowtie \sigma_{S.b > 30}(S)$

Query optimization

3. Search Space: Theoretical space of all possible plans.

System R optimizer \rightarrow 1st implementation of Relational DB.
implements the rules.

$T_1, T_2 \dots T_n$
Left-deep join Tree



Pipelining only works for Nested Loop Join.