Suppose R = 1000 pages bfr = 100 records/block

Number of I/O to find correct page of index 1.2 on avg.
Suppose 1 more I/O needed to get row

Total cost $100000(1+1.2) + 1000 = 221000$

(100000: find all matches. 1000: read R)

Sort merge join – supposed don't have index. idea sort both tables then scan to find matches.

Other operations

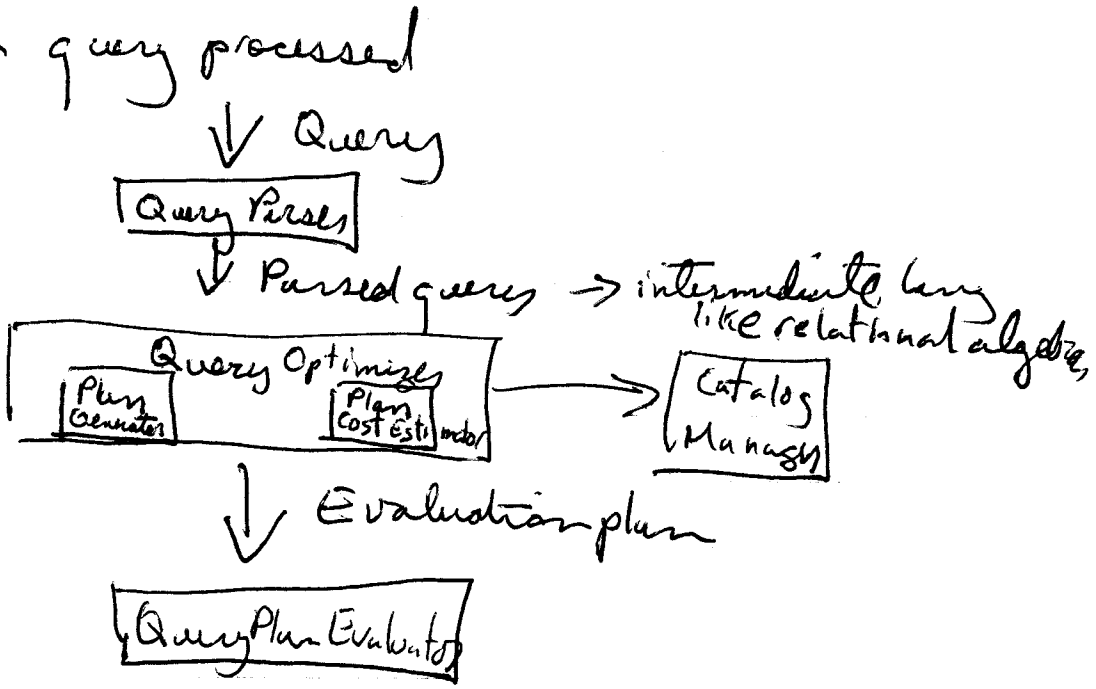
Ex) Union (T) U (S)

Sort both sets. iterate through each set at same time add the smaller item to output if not already output

Ex) Group-by use sorting

Query Optimization

How a query processed

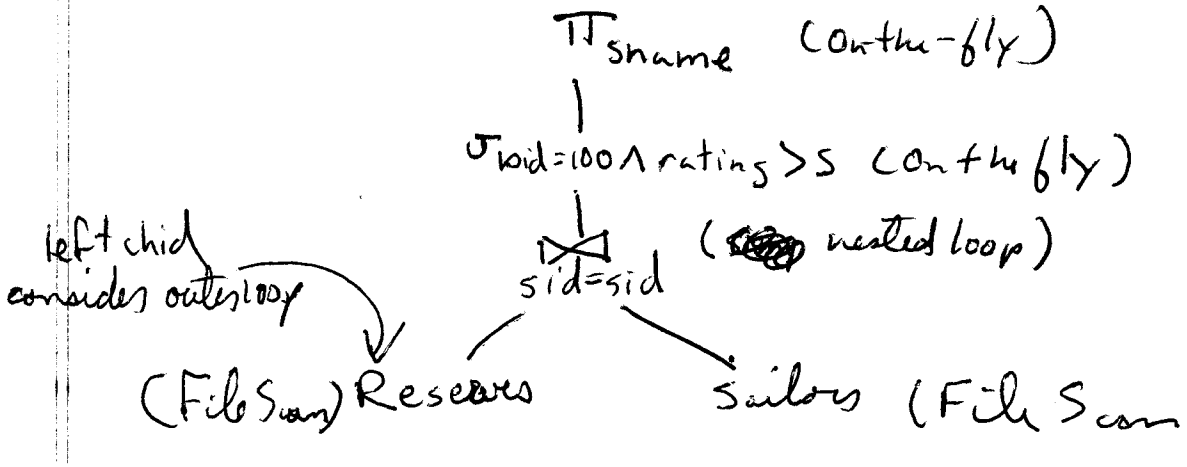


Plans are relational algebra tree w/ additional annotations on the node indicating the access method

Consider

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
  AND R.bid = 100 AND S.rating > 5
```

might correspond to tree



Evaluation

If the result of one operator is sent to another operator in evaluation without creating a temporary table, the evaluation is said to be pipelined. (done on-the-fly)

If a temporary table is used then we say the evaluation is materialized.

Ex) $(A \bowtie B) \bowtie C$

could pipeline results into join w/ C

On the fly operation usually don't increase additional I/O costs

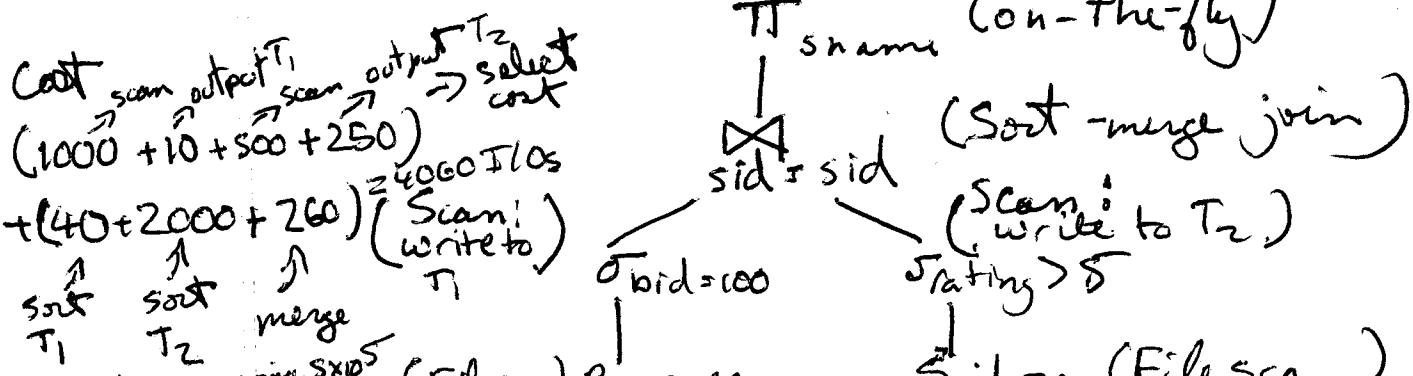
Coming up with alternative plans ...

Usual query optimizer uses a list of standard heuristics, estimates the cost of resulting plans, picks plan with smallest cost.

Some heuristics → Pushing Selections

Since join is expensive want to join as small tables as possible

To do this we push selects passed joins. For example, if we did this to earlier query might get plan:

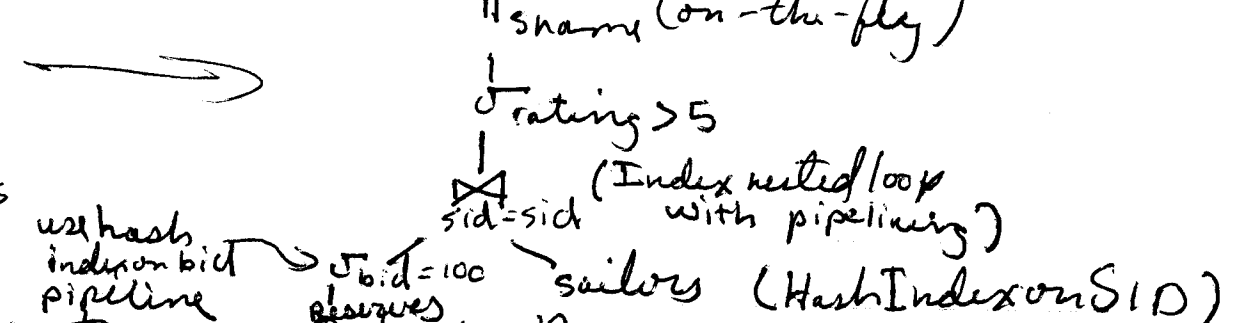


Using Indexes

Optimizer also figures out w/c indexes match given selects and joins and also uses them in coming up w/ new plans. for instance,

Previous query might be written as:

book estimates cost at 1210 I/Os



What a query Optimizer Does

As said before...

The query optimizer takes an original default plan that came from parsing the query.

It considers ~~[illegible]~~ different plans from applying a list of heuristics such as:

- pushing selects and projects below joins (and projects below selects)
- combining selects & cross products into joins.
- reordering joins.

It then sees which indexes match operators in these plans and tweaks some plans.

Then cost estimates of plans done and lowest cost plan selected.

How is reordering of joins done?

Given $A \bowtie B \bowtie C \bowtie D$

better since can pipeline minimizes search space of dynamic programming

would consider → left deep.