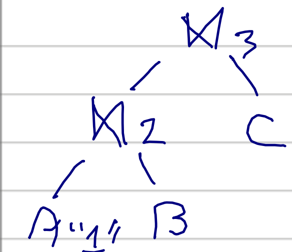
System R

This is a dynamic programming approach to build a best query plan in a bottom-up way.

This relies on the sub-structure optimality property: IF I find a best plan for a sub-query, then I can compose a best over-all plan that uses that sub-query.

We restrict to left-deep trees. E.g.,



IF we are joining three tables, we are figuring out which goes in position A, which in B, and the last in C.

System R works level by level.

Level 1: We find the best "access path" for each "sub-query" — a projection of our query — of just a single table.

By best, we mean the least expensive by I/O.

Level i: We explore taking a best plan as our outer from level i-1 and we join one of the remaining tables (not used in the outer plan) with it as the inner.

We choose the best join algorithm for that particular join by cost.

Note that we need a cardinality estimation of the outer plan to cost each join option.

Caveat: We not only keep a best plan per combination, we also keep a best plan per interesting order. That is, if a plan can output its records in a given sorted order, we keep the best one per such sorted order.

In the last level, we will be composing the best overall plan for the query.

2. (10 points) **Query Planning I.** *Sign up!* [EXERCISE]

Schema:

Student(id, name, major)
Enrol(id, course#, section, term, grade)
FK (id) refs **Student**
FK (course#, section, term) refs **Class**
Class(course#, section, term, instructor, room, time)

Statistics:

- **Student**: 100,000 records on 2,000 pages
  - major: 100 distinct values
- **Enrol**: 4,000,000 records on 40,000 pages
  - course#: 1000, . . ., 4999 (so 4000 values)
- **Class**: 200,000 records on 6,000 pages
  - instructor: 8,000 distinct values

Indexes:

- **Student**:
  - hash index on id (linear hash, 200 data entries per page)
- **Enrol**:
  - clustered tree index on id, course#, section, term (50 data entries per page)
  - unclustered tree index on course#, section, term, id (50 data entries per page)
- **Class**:
  - clustered tree index on course#, section, term (60 data entries per page)
  - unclustered tree index on instructor# (200 data entries per page)

All indexes are of alternative #2. For each tree index, the index pages are 3 deep.

Query:

```
select name, instructor, C.term
    from Student S, Enrol E, Class C
    where S.id = E.id
      and E.course# = C.course# and E.section = C.section
      and E.term = C.term
      and instructor = 'Dogfurry';
```

---

a. (3 points) How many records should the query produce?

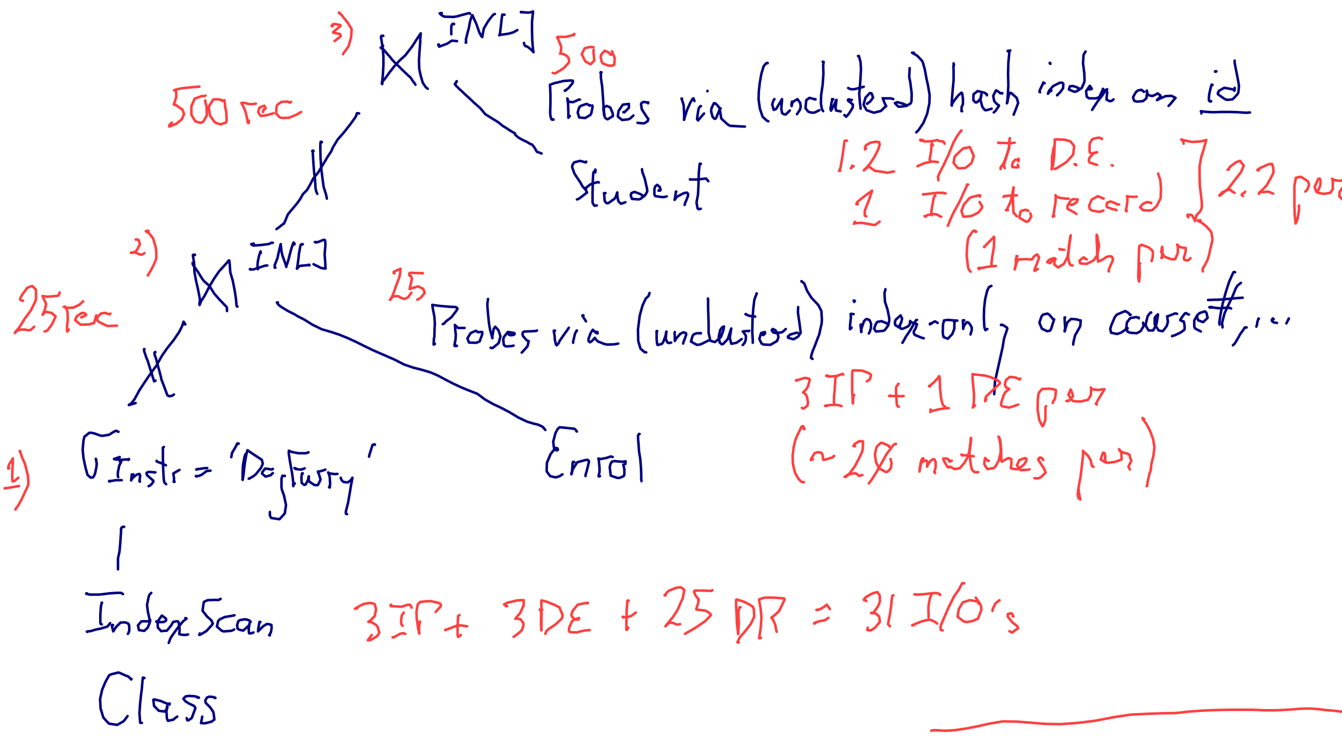
$$\frac{100k \cdot 4M \cdot 200K}{\max(100k, \leq 100k),\ \max(200k, \leq 200k),\ 8K} = \frac{4M}{8K} = 500$$

---

b. (12 points) Devise a good query plan for the query. Show the query tree, *fully* annotated with the chosen algorithms and access paths.

You have an allocation of 20 buffer-pool frames.

Estimate the cost of your plan. For full credit, you should have a plan that costs less than 2,000 I/O's.



1. 31 I/O's
2. 25 x 4 = 100 I/Os
3. 500 x 2.2 = 1100 I/O's

Total: 1231 I/O's

No reading or writing between 1, 2, & 3 as it is all pipelined

3. (15 points) **Query Planning II.** *Of course a course is par for the course.* [EXERCISE]

Schema:

> **Student**(sid, sname, startdate, major, advisor)
> FK (advisor) refs **Prof** (pid)
> **Class**(cid, dept, number, section, term, year, room, time, pid, ta)
> FK (pid) refs **Prof**
> FK (ta) refs **Student** (sid)
> **Enrol**(sid, cid, date, grade)
> FK (sid) refs **Student**
> FK (cid) refs **Class**
> **Prof**(pid, pname, pdept, office)

Assume no attribute is nullable. The attribute pid in **Class** refers to the the professor / instructor for the class. The attribute ta in **Class** refers to the teaching assistant for the class. The attribute advisor in **Student** refers to the student's academic advisor.

Statistics:

- **Student**: 50,000 records on 1,000 pages
  - advisor: 2,500 distinct values
- **Enrol**: 2,000,000 records on 20,000 pages
  - sid: 50,000 distinct values
  - cid: 80,000 distinct values
- **Class**: 80,000 records on 1,600 pages
  - pid: 4,000 distinct values
  - ta: 5,000 distinct values
- **Prof**: 4,000 records on 40 pages

Indexes:

- **Student**:
  - clustered tree index on sid (200 data entries per page)
- **Enrol**:
  - clustered tree index on cid, sid (167 data entries per page)
  - unclustered tree index on sid, cid (167 data entries per page)
- **Class**:
  - clustered tree index on cid (200 data entries per page)
- **Prof**:
  - clustered tree index on pid (200 data entries per page)

All indexes are of alternative #2. For each tree index, the index pages are 3 deep, except for the index on **Prof**.pid which is 2 deep.

```
select sid, sname, dept, number, section, term, year, pid
    from Student S, Enrol E, Class C
    where S.sid = E.sid and E.cid = C.cid
      and S.advisor = C.pid;
```

a. (2 points) Estimate the number of rows the query returns.

SME : FK on E. CME : FK on E. Shortcut :
|Enroll| with any additional reductions.

$$\frac{2M}{max(2.5K, 4K)} = \frac{2000K}{4K} = 500$$

b. (8 points) Devise the best query plan for the query. Show the query tree, *fully* annotated with the chosen algorithms and access paths.
Assume you have an allocation of 50 buffer-pool frames.
Estimate the cost of your plan.

Level one : Best access paths

Sub-query on S : pull all records

S Filescan : 1K

S Indexscan : 1.25 K w/ sorted by sid (interesting order)

reads DE's of clustered (250 pages: 50K/200 per) + 1K of record pages

Sub-query on C : pull all records

C Filescan : 1.6K

C Index Scan : 2K w/ sorted by cid (Using its clustered index as in)

Continued: See attached pages

c. (5 points) Name an additional index that would allow a less expensive query plan than in 3b, and sketch briefly that query plan using the index.

Nah! A bit of a trick question. Here nothing really helps.

Could have index on (sid, advisor, sname) on Student would help marginally; could use index-only scan for S.

4. (10 points) **Query Planning III.** *Down-the-Rabit-Hole Optimization.* [EXERCISE]
   Consider the same tables, statistics, and indexes as in Question 3.

```
select sid, sname, dept, number, section, term, year, pid
    from Student S, Enrol E, Class C
    where S.sid = E.sid and E.cid = C.cid
      and S.sid = C.ta;
```

← Same as E.sid = C.ta

d. (2 points) Estimate the number of rows the query returns.

$$\frac{2M}{\max(5k, 50k)} = \frac{2000}{50} = 40$$

e. (8 points) Devise the best query plan for the query. Show the query tree, *fully* annotated with the chosen algorithms and access paths.
   Assume you have an allocation of 50 buffer-pool frames.
   Estimate the cost of your plan.

Looks the same as Q-II, no?

Level 1 would be!

Level 2, same considerations

But in [CE] we can check E.sid = C.ta on the fly. Cardinality of output: 40 records!!

Level 3 Using [CE] as outer will be huge win. Pick INLJ. Probe S on sid.

Each 3 + 1 + 1 = 5

40 × 5 = 200 for this.

QII Continued.

Level 1 continued.

Still need best access path for Enrol. We note that index-only suffices for E, as both indexes cover the columns of E needed for the query.

E Index-only scan using index #1 : ~12K I/O	(20K/167)	Sorted on cid, sid
E Index-only scan using index #2 : ~12K I/O		sorted on sid, cid

Does not matter that index #2 is unclustered here, since it is index only.
We keep both these access-path plans for E, as they have different interesting orders!

And Filescan of E? Would cost 20K I/O's. We have a better plan than that, so no.

Done w/ Level 1.

Level 2

We need to find best plans for

[E ⋈ S],
[C ⋈ E], and
[C ⋈ S]

To do this, we take a best outer from level 1: [C], [E], or [S] and join w/ a remaining table.

For [ES], there are two possible forms:

[E] ⋈ S or

[S] ⋈ E.

For each form, we're choosing one of the best sub-plans from Level 1 for the outer, and the join algorithm to use.

[S File scan] $\bowtie_{sid}$ E

[BNL]
or [INL] w/ an appropriate index, if one.
or 2 pass SM]
or (2 pass) hash join

Could also consider Merge join
But this needs outer and inner sorted on the join condition. We consider pushing an Ext sort operator over the outer and/or inner, if needed.

And same for

[S index scan] ⋈ sid S

Since we have a second best plan for [S] w/ an interesting order.

First, w/ FileScan on S as outer

BNLJ: Expensive! 50 buffer frames
Read inner $\lceil 1K/48 \rceil$ times!
Well, let's round
$1K/50 = 20$ times
Can just do index-only scan of E as inner (either index): 12K
1K I/O for S
$20 \cdot 12K = 240K$ scanning E on inner
241K

INLJ: Probing inner E 50K times?!
Expensive. 50K even at 1 I/O per!

SMJ: Not enough buffer (50 frames)
12K of E data-entry pages too big

HJ: Yes! 50 Frames suffice.
$3 \times (1K + 12K) = 39K$

MJ: No guarantee S is 100% sorted on sid. (Near 100%, because of clustered index. Still.)
So put sort operator over outer.
1K pages w/ 50 Frames:
Can do in 2 passes, so
$2 \times 2 \times 1K \doteq 4K$
E index-only scan on index #2 already sorted on sid!
So, 4K (sort) + 1K to read outer (Blocking op, so written to disk. Not pipelined!)
+ 12K read inner = 17K

For S File scan ⋈ E, MJ wins.

And w/ interesting order on sid, cid.

What about [S index scan] ⋈ E ?

Do same analysis. Outer cost 1,25K this case, but has interesting order on sid.

BNLJ, INLJ, HJ all possible. Cost out same way; just slightly more expensive because outer is.
SMJ not possible; same as before.

MJ: Don't have to explicitly sort outer as before! Already sorted on sid.
Cost: 1.25K + 12K = 13.25K

Best so Far! But are we done for the best plan for [ES] ?
No! Have to also consider a best plan [E] as outer and S as inner.

The options in this case all work out about the same.

E Index-only-scan on #1 and an index-scan of S on inner makes a MJ with both sorted already, at 12K + 1.25 = 13.25K.
and w/ interesting order sid, cid.

Ties w/ what we found for [S] ⋈ E,
Arbitrarily, keep one of these.

Yay! Now we have a best plan for [ES]. And it has an interesting order on sid, cid.

What is its cardinality? 2M records.
How many pages is that?

Each E Data Entry is 1/167 of a page.
Each S record is 1/50 of a page

$1/50 + 1/167 \approx 1/39$

$1/39 \cdot 2M \doteq 51.2K$ pages

Note we'd not need to keep column sid twice.
And we could drop the RIDs from E's index-only scan.
So these would pack better. But let's go w/ this, for now.

Anything else for Level 2?
Yes, Need best plans for [CE] and for [CS]

[CE] Plays out similarly as for [ES]

Best is MJ using [C index scan] as already sorted on cid and index-only scan on #2 for E on cid, sid.

2K + 12K = 14K. And int. order on cid, sid.
Or, E as outer and C as inner, MJ.
Same as for ES.

[CS] Does it join? Yes. S.adv = C.pid

No useful interesting orders from Level 1.

INLJ : not an option
BNLJ : expensive (Sys R would look.)
HJ : 7.8 K I/O. 3×(1K + 1.6K)

SMJ : 7.8 K I/O. But, Technically, not enough buffer pool ...
MJ : Sort each : can do in two passes each
2×2×1K + 2×2×1.6K = 10.4K
Then read to merge sorted : 1K + 1.6K
= 13K
Keep. It is an interesting order. But on pid.
(We know it cannot be useful)

Size of [CS] : $\frac{50k \cdot 80k}{max(2.5k, 4k)} = 2M$

#pages? : $\left(\frac{1}{50} + \frac{1}{50}\right) \cdot 2M \doteq 40K$

Level 3 Finally!

Take a best plan as outer from level 2, join w/ remaining.

[ES] ⋈ C,

[CE] ⋈ S,

or

[CS] ⋈ E.

In each case here, we've a best plan for [ES], [CS], and two for [CS]

Sys R would consider each, in turn, & the join algorithm options.

INLJ's : Probing inner 1M or 2M times? Will be too expensive

BNLJ's : Outers are all large. Many scans of inner. Will be too expensive.

SMJ : Not enough Buffer Frames...

MJ : No outer is sorted as we need... Expensive to sort 40K or 50K in 50 Frames!

HJ : Yes. Inner in C or S case small enough. Since size of CE and ES same, go w/ [CE], leaving a smaller inner

$\boxed{CE} \bowtie^{HJ} S$

$$3 \times (51.2^{K} + 1^{K}) = 156.8K$$

$\boxed{1\ CE}$ cost 14K

Total: 170.8 K

All this work for 500 records!