2015/09/07 ... Possibly Pale Blue

A table T(PK, A, B, C, RefToIDs5) is primarily stored as entry-sequenced, with 40K-tuples, in 1K-blocks. Table S, which keeps table S(ID, X, Y) contains 1M tuples in a primary-hash-based storage, indexed by the primary key, with 100K buckets and very sparse, virtually free from overflow chains.

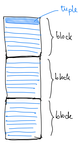
Knowing that PK<1000 for 2% of the tuples in T, that A is a unique attribute, and that val(B) = 125 (homogeneously distributed), estimate the execution cost of the query below, in the following three scenarios:

1. No secondary indexes are **available**

2. There is a B+ index with F = 200 for T, on the primary **key**

3. There is also another B+ index on attribute B, with depth 3 (a root, an intermediate level, and 1.25K leaf nodes).

```
select *
from S join T on RefToIDs5 = ID
where PK < 1000 and B = "pale blue" and A <= 13472
```



e.g. 40 tuples / block

if we want 15 tuples we have to read 2 blocks.

ENTRY SEQUENCED

T

PK	A	B	C	RPT

No order

S

ID	X	Y

HASH-BASED ACCESS

PK < 1000

* 40K x 2% = 800 tuples

B = "pale blue"

* 40K / 125 = 320 tuples

* TOT = 1.12K tuples

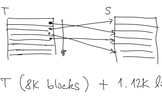
WORST CASE: largest result size

(1) Nested-Loop ?

Not the best (DISCUSS)

We have a hash primary structure on S

⇒ We can scan T and lookup in S



Scan T (8K blocks) + 1.12K lookups on S = 9.12K

⚠ We have to scan all T (it's unordered)

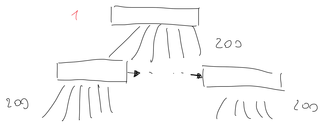
⚠ We select tuples of T that match the WHERE

⚠ For each selected tuple we join it with S (using hash ⇒ 1 access!!)

(2) B+, F=200 for T on primary-key

⚠ B+ tree can be used to access RANGE values

PK < 1000 → easy



$\underbrace{(1 + 800/F)}_{\text{NODES}} + \underbrace{800}_{\text{ACCESS TO BLOCKS}} = 805$

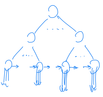
BUT

NO INFO ON "PALE BLUE" FOR PK ≥ 1000

⇒ Previous plan is still the best

(3) Another B+ Tree on B (depth 3)

1.25K leaf nodes



1.25K / 125 = 10 blocks/color

(a) Use B+ on PK as the previous one

(b) look for "pale blue" in the new tree

then ELIMINATE DUPLICATES

then lookup in S

(2) # of nodes

$\underbrace{1 + 4 + 800}_{\text{SAME AS PREVIOUS}} + \underbrace{1 + 10 + 320}_{\text{\# of tuples}} + 1.12K = 2.26K$

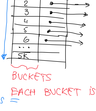
A table Role(Actor, Movie, Character) records 400K roles played by Hollywood actors in over many decades. Estimate the execution cost (under reasonable assumptions) of the following query in the scenarios listed below. Please briefly describe the considered query plan in each scenario.

```
select Actor, Movie, count(*) as NumberOfCharacters
from Role
group by Actor, Movie      // Extracts actors playing 3+ roles in the same movie
having count(*) > 2
```

1. The table is primarily stored in 16K blocks, with tuples in no particular order. There is also a hash based secondary index with Movie as key, with 5K buckets of 1 block each (val(Movie)=20K, val(Actor)=25K).
2. The table is primarily stored in 16K blocks, with tuples sequentially ordered according to the Movie attribute (as they are sequentially appended as soon as new movies are released), and there are no secondary access structures.
3. The table is primarily stored as in case 1, but the secondary structure, instead of being a hash, is a B+ tree with two attributes as key (Actor, Movie) – i.e., the key is composed of the two attributes, in this order. The tree has depth 3 (a root, an intermediate level, and 3.5K leaf nodes).

(1) Hash on Movie

* Movies are already grouped by the hash function.
* Every h(M) is a different Movie



BUCKETS

EACH BUCKET IS 1 BLOCK

400K tuples / 20K Movies = 20 pointers

⇒ 20 pointers per Movie

For each of the 20K Movies we follow 20 pointers

5K + 20K x 20 = 405K

INDEX BLOCKS

(2) * SEQUENTIALLY ORDERED (Movie)

* NO INDEXES

400K / 16K = 25 ROLES/BLOCK

400K / 20K = 20 ROLES/MOVIE

} Roles in a few adjacent blocks

⇒ SINGLE SCAN

⇒ 16K accesses (+ possibly little overhead).

(3) * IF LEAF NODES CONTAIN AS MANY POINTERS TO THE BLOCKS AS THE TUPLES OF PRIMARY STORAGE WE DO NOT NEED TO RETRIEVE THE TUPLES!!

* LEAF NODES ALREADY CONTAIN GROUPED VALUES!

* WE CAN COUNT THE POINTERS!

DEPTH 3

3.5K LEAF NODES

IF 2 TUPLES WITH SAME ACTOR AND SAME MOVIE ARE IN THE SAME BLOCK

⇒ WE HAVE TO SCAN THE ENTIRE STRUCTURE ⇒ 400K ACCESSES

(SCENARIO 1)