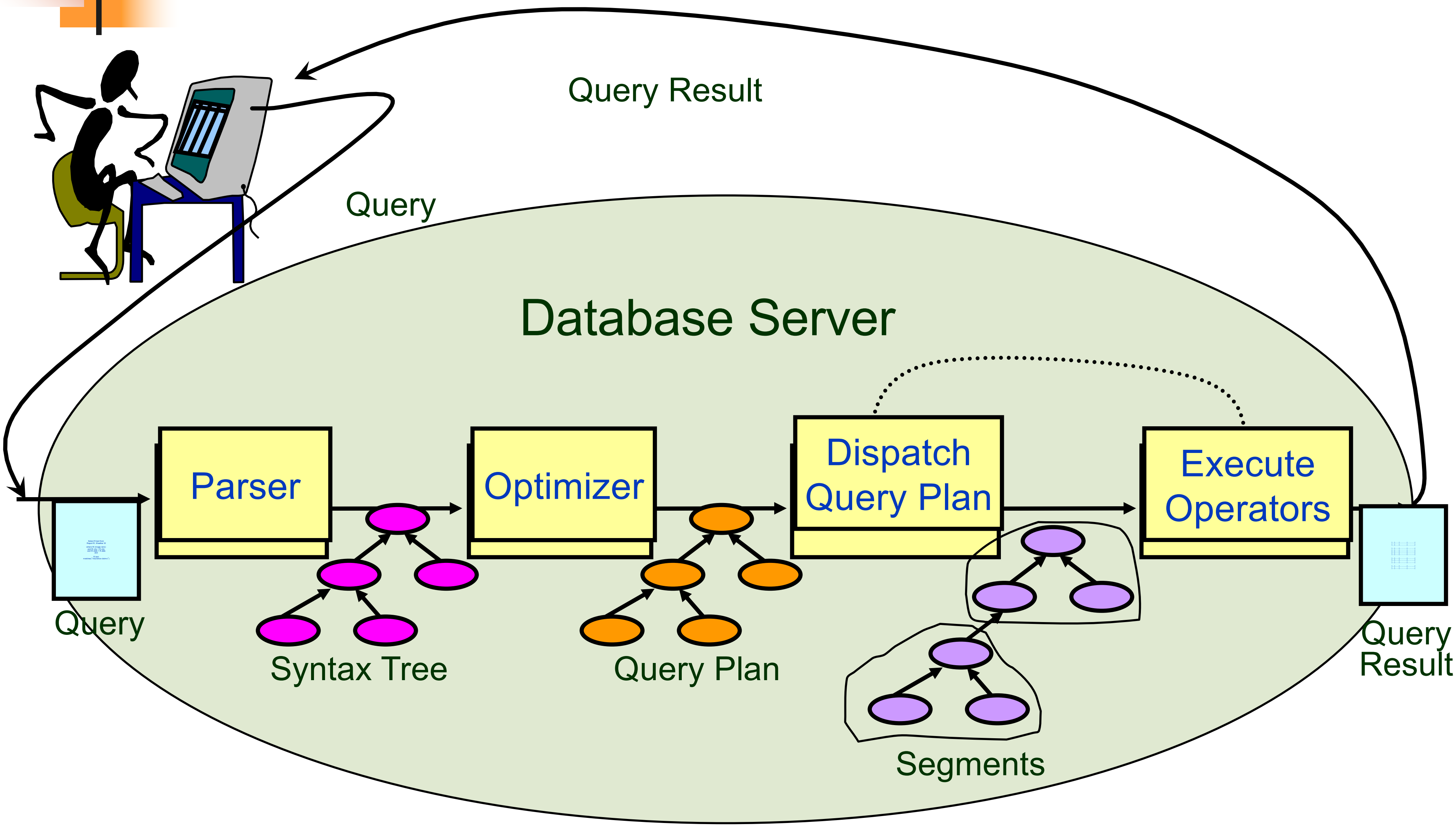
Evaluation of Relational Operations

Chapter 12 and 14

Query Execution Life-Cycle



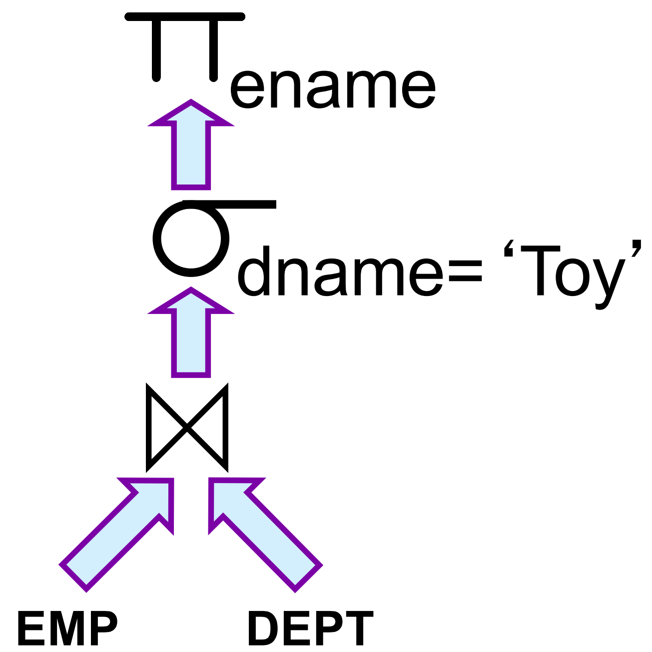
Operator Evaluation

- How to implement common operators?
  - Selection
  - Projection (optional DISTINCT)
  - Join
  - Set Difference
  - Union
  - Aggregate operators (SUM, MIN, MAX, AVG)
  - GROUP BY
- Next week – How to choose a plan?
- Catalogs consulted to parse, optimize, execute plan

Query Evaluation Plan

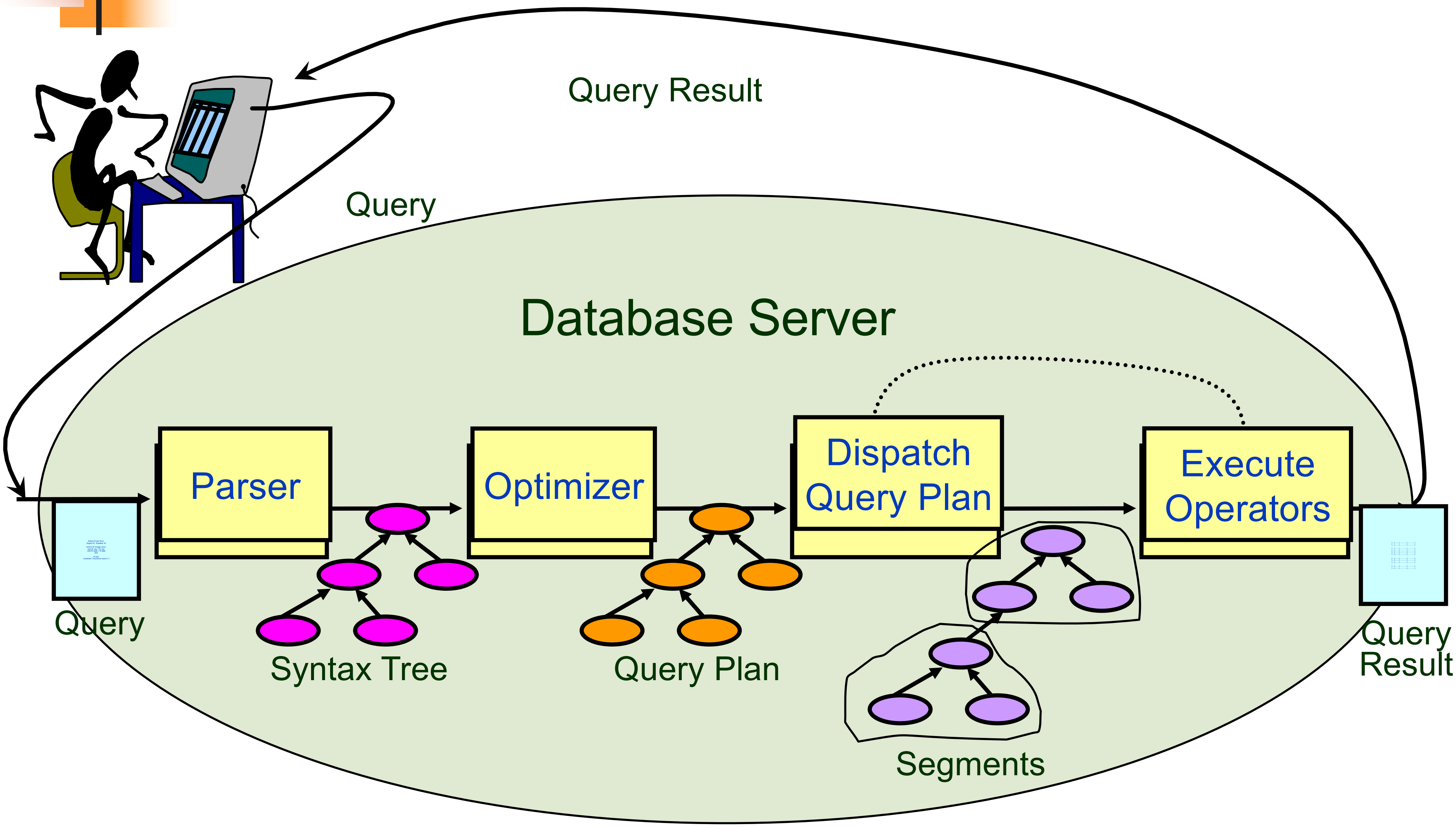
EMP (ssn, ename, addr, sal, did)
DEPT (did, dname, floor, mgr)

SELECT E.ename
FROM Emp E, Dept D
WHERE D.dname = 'Toy'
AND D.did = E.did



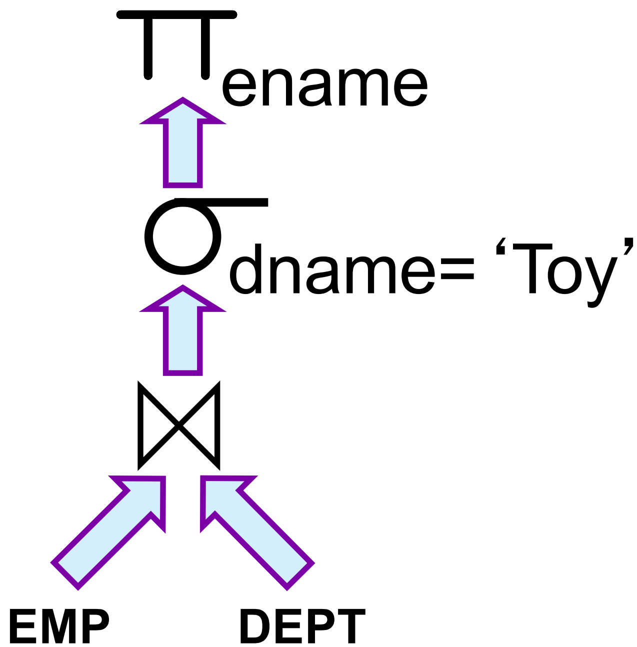
Query Optimizer selects the evaluation plan

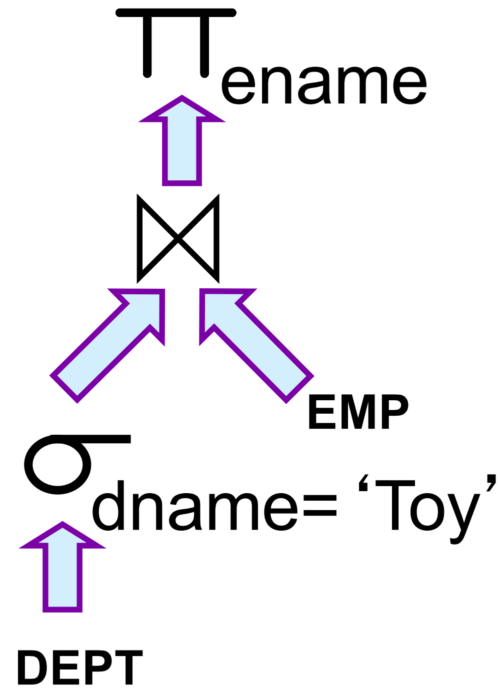
Query Execution Life-Cycle



Query Optimization

- May modify the query plan.





Also, different algorithms for SELECT, JOIN, and PROJECT based on
whether an index is available, table sizes, etc.

Selection (on one table)

- An access path defines the strategy to use to do a *selection* on a table, possibly utilizing an index
- Example of a selection condition
  - a predicate: gpa > 3.0 and age = 21
- Examples of access paths
  - File scan
  - Index that *matches* a selection in the query. Examples:
    - B+tree index on the <gpa, age> attributes
    - B+tree index on gpa
    - B+tree index on age
    - Hash index on age

Selection

- Where R.a *op* value
- Options:
  - Heap file — Cost: O(N)
  - Sorted File — Cost: $O(\log_2 N)$ + …
  - Index
    - Hash — Cost: O(1) + …
    - B+Tree: Clustered/Unclustered — Cost: $O(\log_F N)$ + …

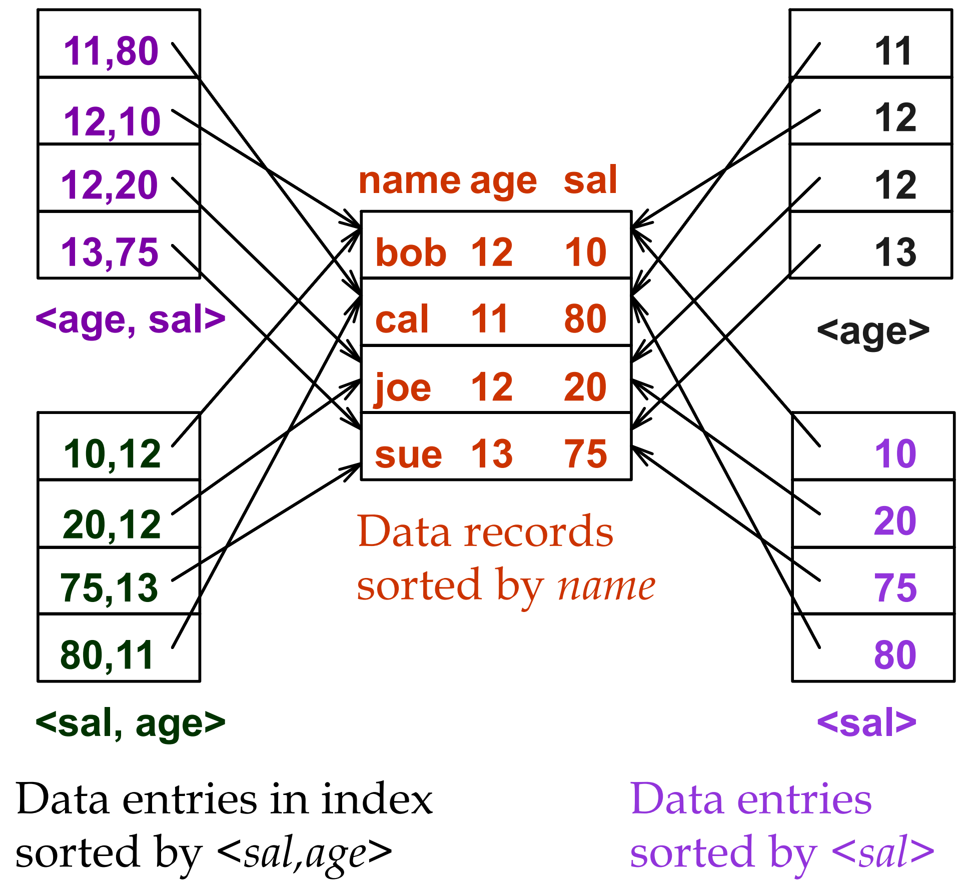
Composite Search keys

- Index on <age>: search key is a single-attribute age
- Index on <name, age>: search key is *composite* (name, age) pair.
  - For B+-tree, name is the primary comparison attribute and age matters only when names are equal
  - For hash-index, h((name, age)) used – both name and age are needed to hash.

Indexes with Composite Search Keys

- Composite Search Keys: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. e.g. wrt <sal,age> index:
    - age=12 and sal =75
  - Range query: Some field value is not a constant. e.g.:
    - age =12; or age=12 and sal > 10
- Data entries in index sorted by search key to support range queries.

Examples of composite key indexes using lexicographic order



Index Matching

- When can we use an index to evaluate a selection predicate?
- An index *matches* a predicate if index can be used to evaluate the predicate

Exercise: Index Matching

- Index on <a, b, c>

	Tree Idx	Hash Idx
a=5 and b= 3?	yes	no!
a > 5 and b < 3	yes	no!
b=3	no!	no!
a=7 and b=5 and c=4 and **d>4**	yes	yes
a=7 and c=5	yes	no!

- Index matches (part of) a predicate if
  - Conjunction of terms involving only attributes (no disjunctions)
  - Hash: only equality operation, predicate has all index attributes.
  - Tree: Attributes are a prefix of the search key, any ops.

Index Selectivity

- To retrieve Emp records with *age*=30 AND *sal*=4000, an index on <*age, sal*> would be better than an index on *age* or an index on *sal*.
  - <age, sal> is more *selective* than just <age> or just <sal>
  - It identifies fewer spurious records that will later be rejected
- If condition is: *age*=80 AND 3000 < *sal* <20,000:
  - <*age*> index much better than <*sal*> index!
- *Ideally, the more selective index preferred*

Index Matching

- Predicate could match more than 1 index
- Hash index on <a, b> and B+tree index on

<a, c>

Predicate: a=7 and b=5 and c=4. Which index?

  - Option 0: Neither. Simply use file scan
  - Option 1: More selective one. Then, scan among the selected records.
  - Option2: Use both!
  - Algorithm: Intersect rid sets.
    - Sort rids, retrieve rids in both sets.

Quiz: Selection

- Hash index on <a> and Hash index on <b>
  - a=7 **or** b>5 Which index?
  - Neither! File scan required for b>5
- Hash index on <a> and B+-tree on <b>
  - a=7 **or** b>5 Which index?
  - Option 1: Neither
  - Option 2: Use both! Fetch rids and union
    - Note: Option 1 could be better sometimes. (When?)
- Hash index on <a> and B+-tree on <b>
  - (a=7 **or** c>5) and b > 5 Which index?
  - B+-tree (high selectivity) or File Scan (poor selectivity)

When to use a B+tree index

- Consider
  - A relation with 1M tuples
  - 100 tuples on a page
  - 500 (key, rid) pairs on a page

data pages
= 1M/100 = 10K pages
leaf idx pgs
= 1M / (500 * 0.67)
~ 3K pages

	1% Selection	10% Selection
Clustered	~ 30 + 100	~ 300 + 1000
Non-Clustered	~ 30 + 10,000	~ 300 + 100,000
NC + Sort Rids	~ 30 + (~ 10,000)	~ 300 + (~ 10,000)

⇨ Choice of Index access plan, consider:
1. Index Selectivity **2. Clustering**
⇨ Similar consideration for hash-based indices

System Catalogs

- To help optimize queries, the system keeps information on each relation
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each view:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

Example catalog: Attribute Catalog

attrName	relName	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Catalogs are themselves stored as relations

Basic Join Strategies

Join Operator

```
SELECT  *
FROM    Reserves R, Sailors S
WHERE   R.sid = S.sid
```

- Commercial systems spend a lot of effort optimizing equality joins
  - Why is this important?
  - What is the major source of performance cost when joining two (large) relations?
- Cost Metric: # of I/Os
  - (We'll ignore final output cost)

Join Operator

- Many different ways of evaluating joins

Sailors

sid	name
1	Lucky
2	Rusty
3	Bob
4	Fred

$$\bowtie_{sid = sid}$$

Reserves

sid	bid
1	100
1	200
3	300
4	200

How would you evaluate this join in memory?

Simple Nested Loops Join

foreach tuple r in R do
    foreach tuple s in S do
        if r.sid == s.sid then add <r, s> to result

- Cost Model
  - ||R|| = # tuples in R
  - |R| = # pages in R
- How many I/Os ?

  |R| + ||R|| * |S|

Slightly different notation from textbook!

Page-Oriented Nested Loops

- Page-oriented Nested Loops join:
  - For each page of R, get each page of S, and join
  - How many I/Os?

    |R| + |R| * |S|
  - If S is the outer, then |S| + |R| * |S|

How many buffer pages does this use?

Block Nested Loops

- Can we exploit available memory?
- Suppose we have B buffer pages available.
- Use B-2 pages to hold a block of **outer** R.

```
foreach block of B-2 pages of R do
    foreach page of S do
        foreach r in the B-2 R pages do
            foreach tuple s in the S page do
                if r.sid == s.sid  then add <r, s> to result
```

- Cost: $|R| + |S| * \left\lceil \frac{|R|}{B-2} \right\rceil$

What is the cost if the smaller relation fits entirely in memory?

PNL vs BNL

- |R| = 128 |S| = 64 B = 8
  tuples/page for both S and R = 10
- Page NL
  - Scan outer: 64
  - Join: 64 * 128 = 8192
  - TOTAL: 8256

In PNL, which rel. should be the outer?

- Block NL
  - Scan outer: 64
  - Join: ⌈64/6⌉* 128 = 1408
  - TOTAL: 1472

What about simple nested loops?

Announcements

- Optional Exercises:
  - 12.1 (1-4), 12.3, 12.5