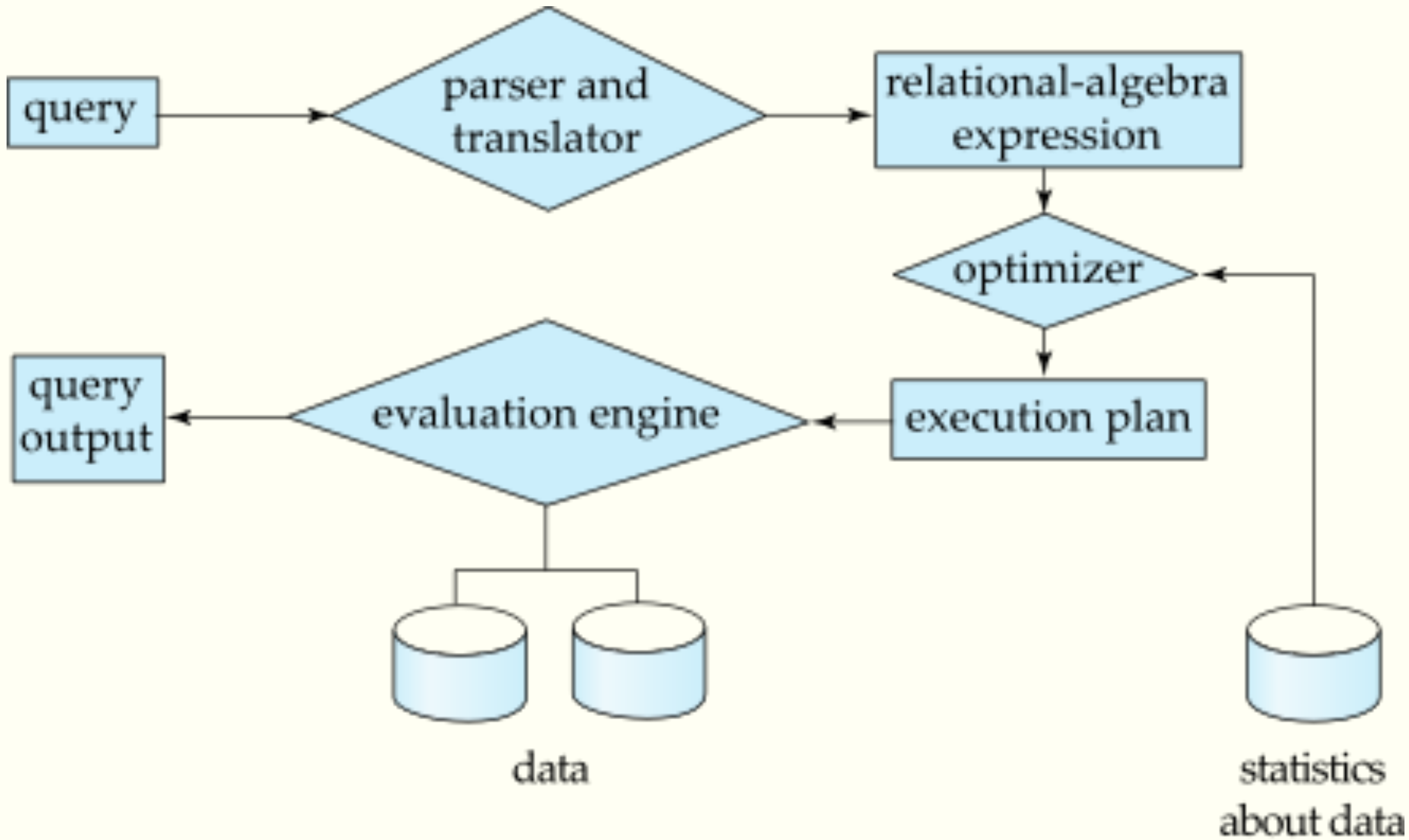
CptS 415 Big Data

Query Language

Srini Badri

Acknowledgement: Tinghui Wang

Basic Steps in Query Processing



Basic Steps in Query Processing

- Parsing and Translation
  - Translate the query into its internal form
  - For RDBMS and SQL DB: Relational Algebra
  - Parser Checks syntax, verifies relations
- Optimization
  - Generate query (evaluation) plan from relational algebra
- Evaluation
  - The query execution engine takes a query evaluation plan, executes that plan and returns the answers to the query

Example Query

- Query is parsed, generally broken into predicates, then converted into a logical query plan used by the optimizer

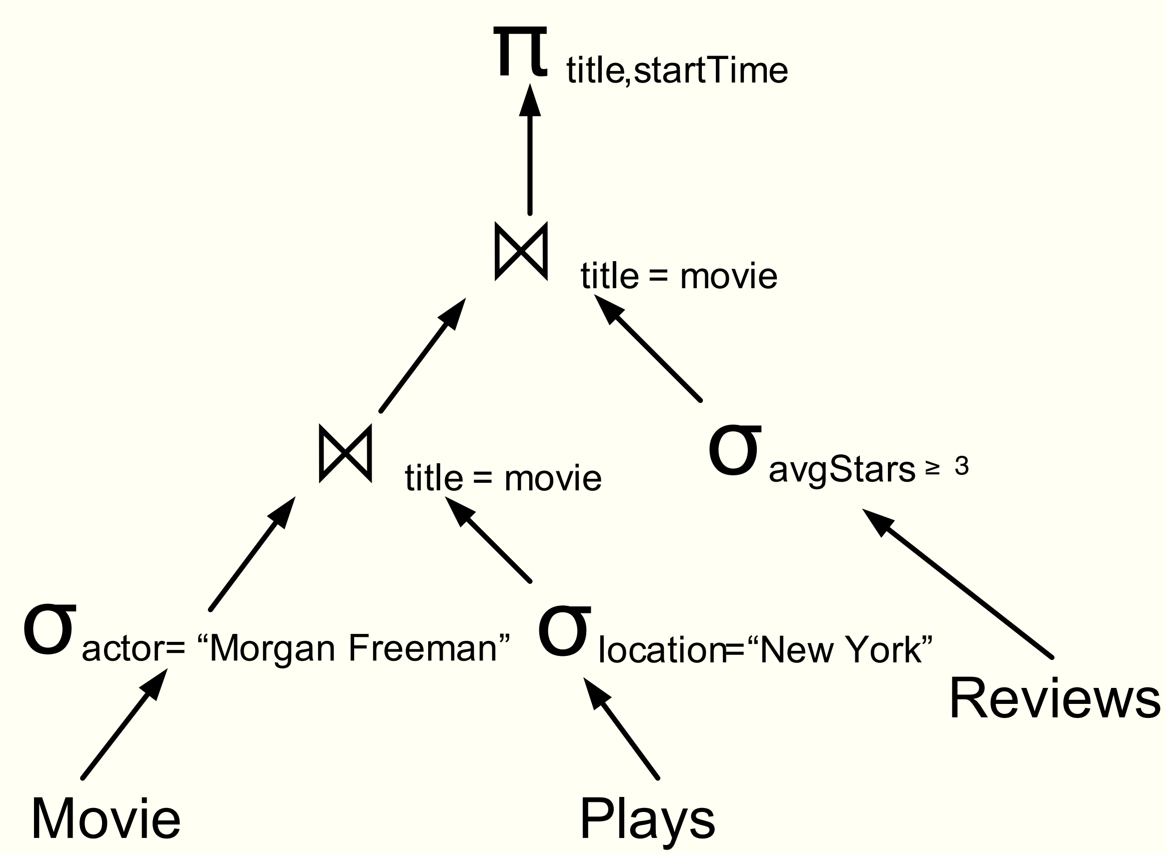
```
SELECT title, startTime
FROM Movie M, Plays P, Reviews R
WHERE M.title = P.movie AND M.title = R.movie AND
    P.location ="New York " AND M.actor = "Morgan
Freeman" AND
    R.avgStars >= 3
```

Example Query Logic Plan

```sql
SELECT title, startTime
FROM Movie M, Plays P, Revi
ews R
WHERE M.title = P.movie AND

    M.title = R.movie AND
    P.location ="New York "
 AND

M.actor = "Morgan Freeman"
AND
    R.avgStars >= 3
```



Query Optimization

- A relational algebra expression may have many equivalent expressions

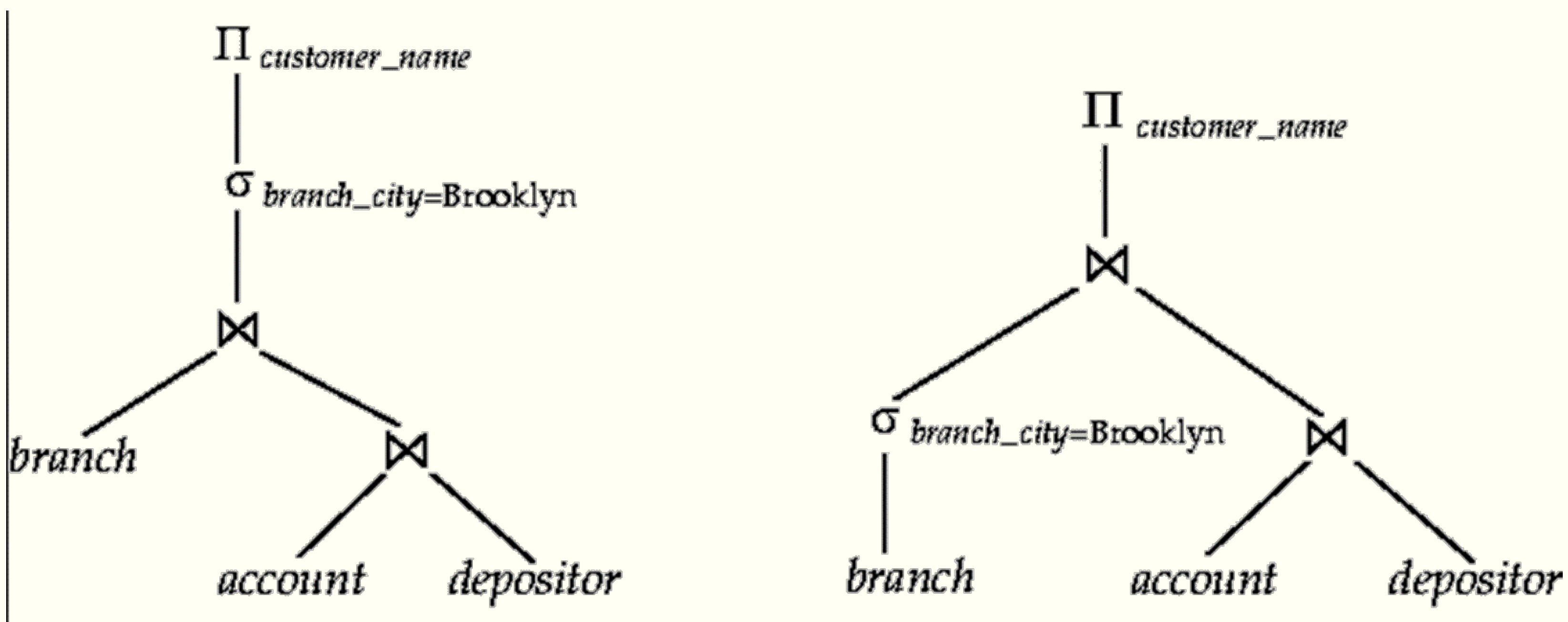
$$\sigma_{salary<75000}\left(\pi_{sallary}(Instructor)\right)$$

$$\pi_{sallary}\left(\sigma_{salary<75000}(Instructor)\right)$$

- Each relational algebra operation can be evaluated using one of several different algorithms
- Annotated expression specifying detailed evaluation strategy is called an evaluation-plan.
  - can use an index on salary to find instructors with salary < 75000,
  - or can perform complete relation scan and discard instructors with salary >= 75000

Example: Query Optimization

- Query: find the names of all customers who have an account at any branch located in Brooklyn
- Relational expression:



(a) Initial expression tree (b) Transformed expression tree

Query Optimization

- Goal:
  - Compare all equivalent query expressions and their low-level implementations (operators)
- Can be divided into
  - Plan enumeration (“search”)
  - Cost estimation

Foundations of Query Plan Enumeration

- Exploit properties of the relational algebra to find equivalent expressions

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

$$\sigma_\alpha(R \bowtie S) = \left(\sigma_\alpha(R) \bowtie S\right)$$

- Assume selection, projection are always done as early as possible

- Joins (generally) satisfy principle of optimality
  - Best way to compute $R \bowtie S \bowtie T$ takes advantage of the best way of doing a two way join, followed by one additional join

Enumerating Plans

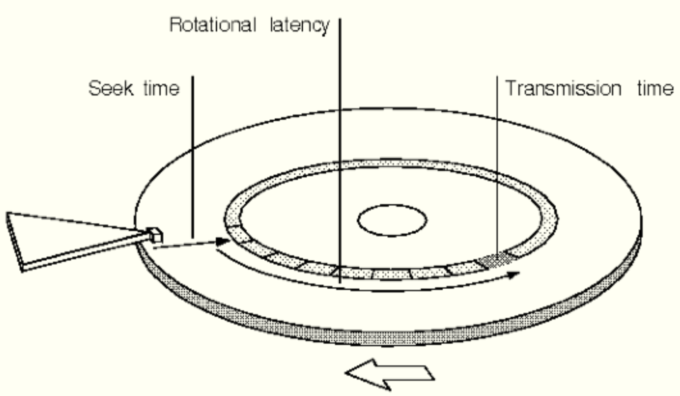
- Can formulate as a dynamic programming problem
- 1. Base case:
  - Consider all possible ways of accessing the base table, with all selections & projections pushed down
- 2. Recursive case (i=2… number of joins + 1):
  - Explore all ways to join results (i-1) tables, with one additional table
  - Common heuristics: only considers linear plans
- 3. Then repeat process for all Cartesian products
- 4. Apply grouping and aggregation

Cost Analysis

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
  - Disk access, CPU or network communication
- Disk access is the predominant cost (I/O) measured by
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
  - Cost to write a block is greater than cost to read a block
  - Data is read back after being written to ensure the write is successful

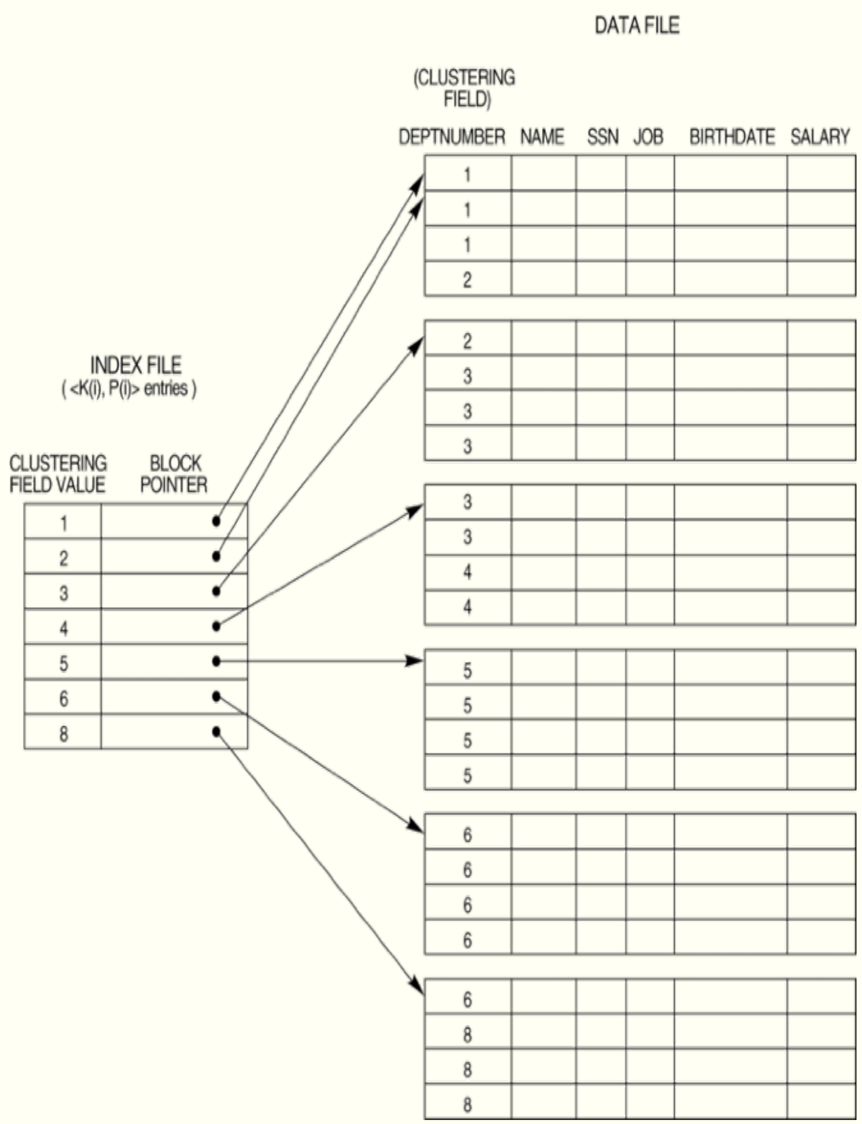
Measures of Query Cost

- I/O Cost
  - Use the number of block transfers from disk and the number of seeks as the cost measures.
  - t_τ: Time to transfer one block
  - t_S: Time of one seek
  - Cost of b blocks of transfer plus S seeks



- CPU cost
  - For main-memory algorithms
- Index are often used to reduce cost

Single-level Index: Primary and Secondary



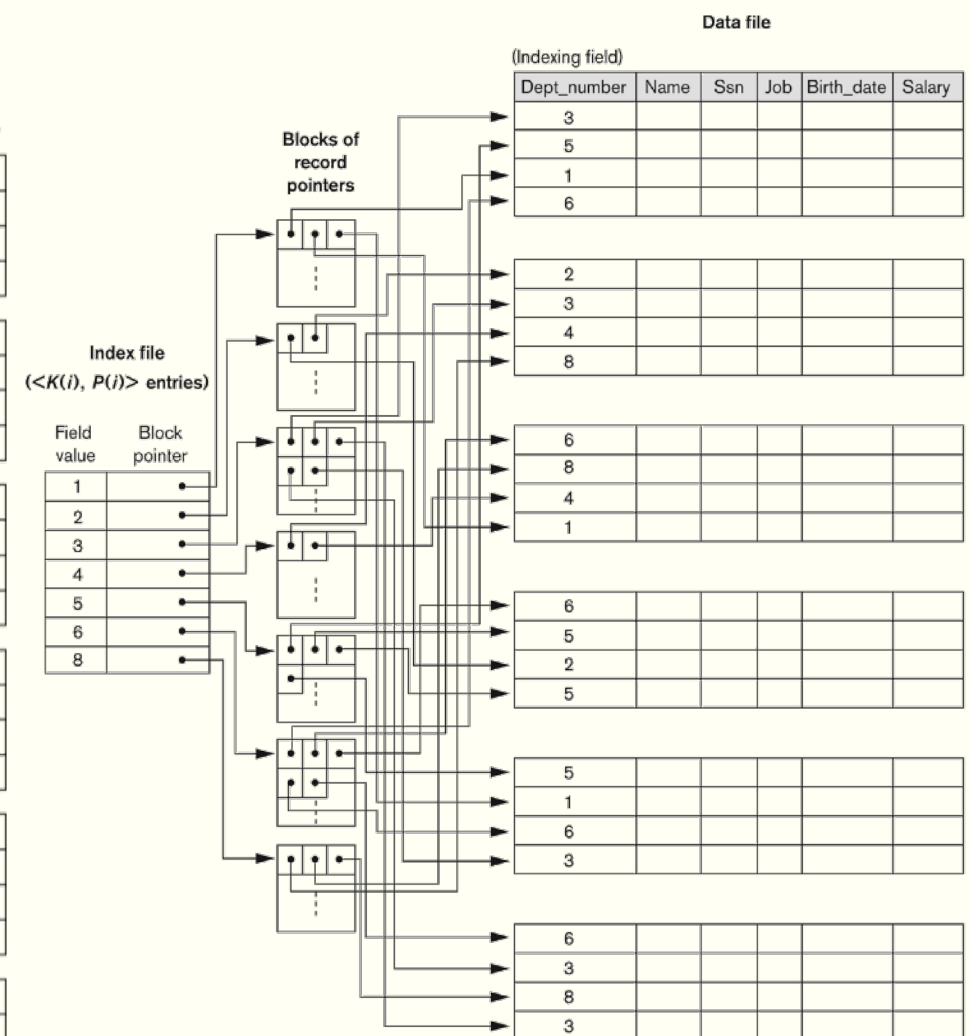
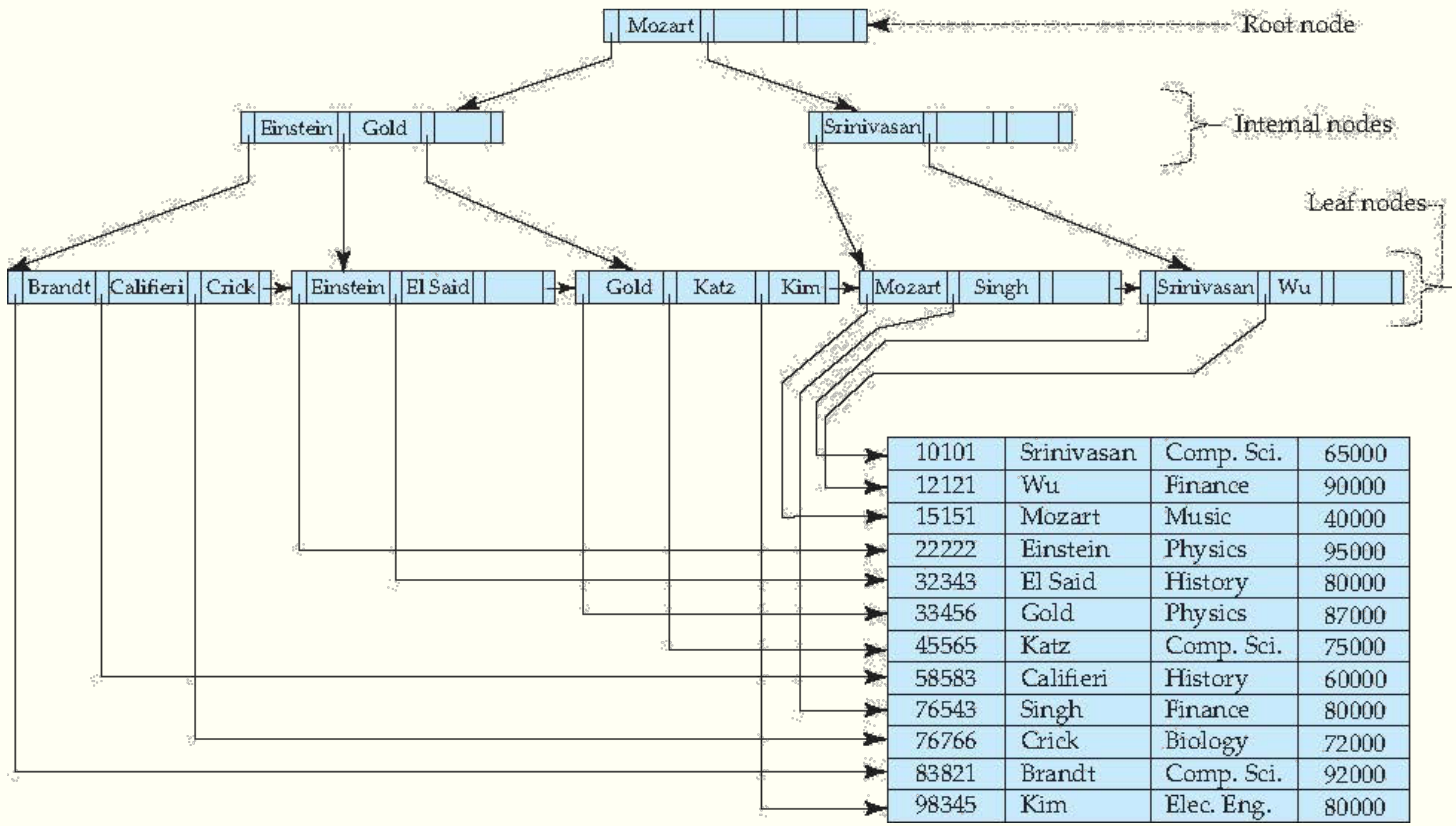


Figure 14.5
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

Multi-level Index: B+ Tree



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Catalog information for cost estimation

- NR:
  - # of tuples in a relation R
- BR:
  - # of blocks that contain tuples of relation R
- SR:
  - size of tuple of R
- FR:
  - blocking factor; # of tuples from R that fit into one block: FR=NR/BR
- V(A, R):
  - # of distinct value for attribute A in R.
- Sc(A,R):
  - selectivity of attribute A = average number of tuples of R that satisfy an equality condition on A; Sc(A,R)=NR/V(A,R)

Query Plan Cost Estimation

- For each expression, predict the cost and output size given what we know about the inputs
- Requires significant information
  - Cost formula for each algorithm, in terms of disk I/O, CPU speed, …
  - Calibration parameters for host machine performance
  - Information about the distributions of join and grouping columns

Selection

- File Scan:
  - A1 Linear scan: Scan each file blocks and test, Cost: $\mathrm{BR} * t_T + t_S$
- Index
  - A2 Primary Index, equality on key
    - Retrieve a single record that satisfies the corresponding equality condition
    - $(h_i + 1) * (t_T + t_S)$
  - A3 Primary index, equality on Nonkey
    - Retrieve multiple records
    - Assume records will be on consecutive blocks, b = number of blocks containing matching records

$$h_i(t_T + t_S) + b * t_T = h_i(t_T + t_S) + \frac{Sc(A,R)}{FR} * t_T$$

NR: # of tuples
BR: # of blocks
SR: size of tuple of R
FR: block size
Sc(A,R): selectivity

Selection

- Index
  - A4 Secondary Index, equality on NonKey
    - Retrieve a single record if the search key is a candidate key

$$Cost = (hi + 1) * (tT + tS)$$

    - Retrieve multiple records if the search key is not a candidate key
      - Each of n matching records may be on a different block

$$Cost = (h_i + n) * (tT + tS)$$

Selections Involving Comparisons

- Comparison:
  - A linear file scan
  - Or single-level index as follows
- A5 (primary index, comparison). (Relation is sorted on A)
  - For $\sigma_{A>=V}(R)$ use index to find first tuple >= V and scan relation sequentially from there
  - For $\sigma_{A<=V}(R)$ just scan relation sequentially till first tuple $> V$; do not use index
- A6 (secondary index, comparison).
  - scan index sequentially to find pointers to records
  - retrieve records that are pointed to
    - requires an I/O for each record
    - Linear file scan may be cheaper

Implementation of Complex Selection

- Conjunction: $\sigma_{\theta 1} \wedge_{\theta 2} \wedge \ldots_{\theta n}(R)$
- A7 (conjunctive selection using one index).
  - Select a combination of θ_i that results in the least cost for $\sigma_{\theta i}(R)$.
  - Test other conditions on tuple after fetching it into memory buffer.
- A8 (conjunctive selection using composite index).
  - Use appropriate composite (multiple-key) index if available.
- A9 (conjunctive selection by intersection of identifiers).
  - Requires indices with record pointers.
  - Use corresponding index for each condition and take intersection of all the obtained sets of record pointers.
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory.

Algorithms for Complex Selection

- Disjunction: $\sigma_{\theta 1} \vee {}_{\theta 2} \vee \ldots {}_{\theta n}(R)$.
- A10 (disjunctive selection by union of identifiers).
  - Applicable if all conditions have available indices.
  - Otherwise use linear scan.
  - Use corresponding index for each condition and take union of all the obtained sets of record pointers.
  - Then fetch records from file

- Negation: $\sigma_{\neg\theta}(R)$
  - Use linear scan on file
    - If very few records satisfy $\neg\theta$, and an index is applicable to θ
    - Find satisfying records using index and fetch from file

Sorting

- We may build an index on the relation, and then use the index to read the relation in sorted order.
  - May lead to one disk block access for each tuple.
- For relations that fit in memory, techniques like quicksort can be used.
- For relations that don't fit in memory, external sort-merge is a good choice.

External Merge Sort

- Let *M* denote memory size (in pages).
- Create sorted runs.
  - Let i be 0 initially.
  - Repeatedly do the following till the end of the relation:
    (a) Read M blocks of relation into memory
    (b) Sort the in-memory blocks
    (c) Write sorted data to run file Ri; increment i.
  - Let the final value of i be N
- Merge the runs (next slide)…..

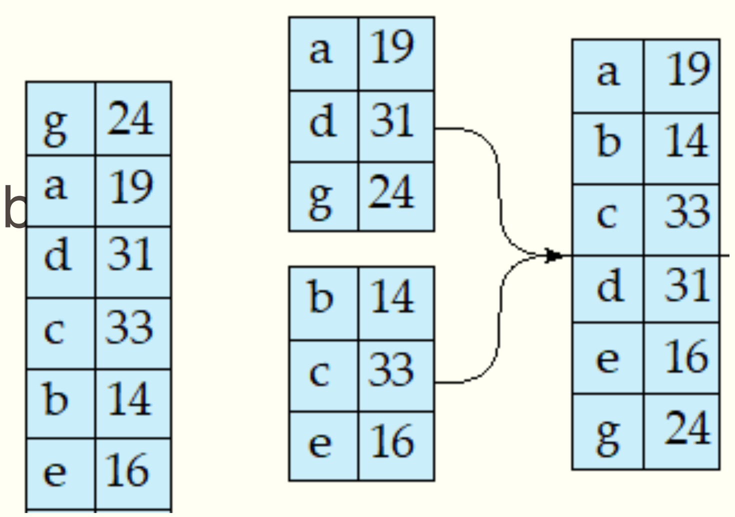
g	24
a	19
d	31
c	33
b	14
e	16

a	19
d	31
g	24

b	14
c	33
e	16

External Merge Sort

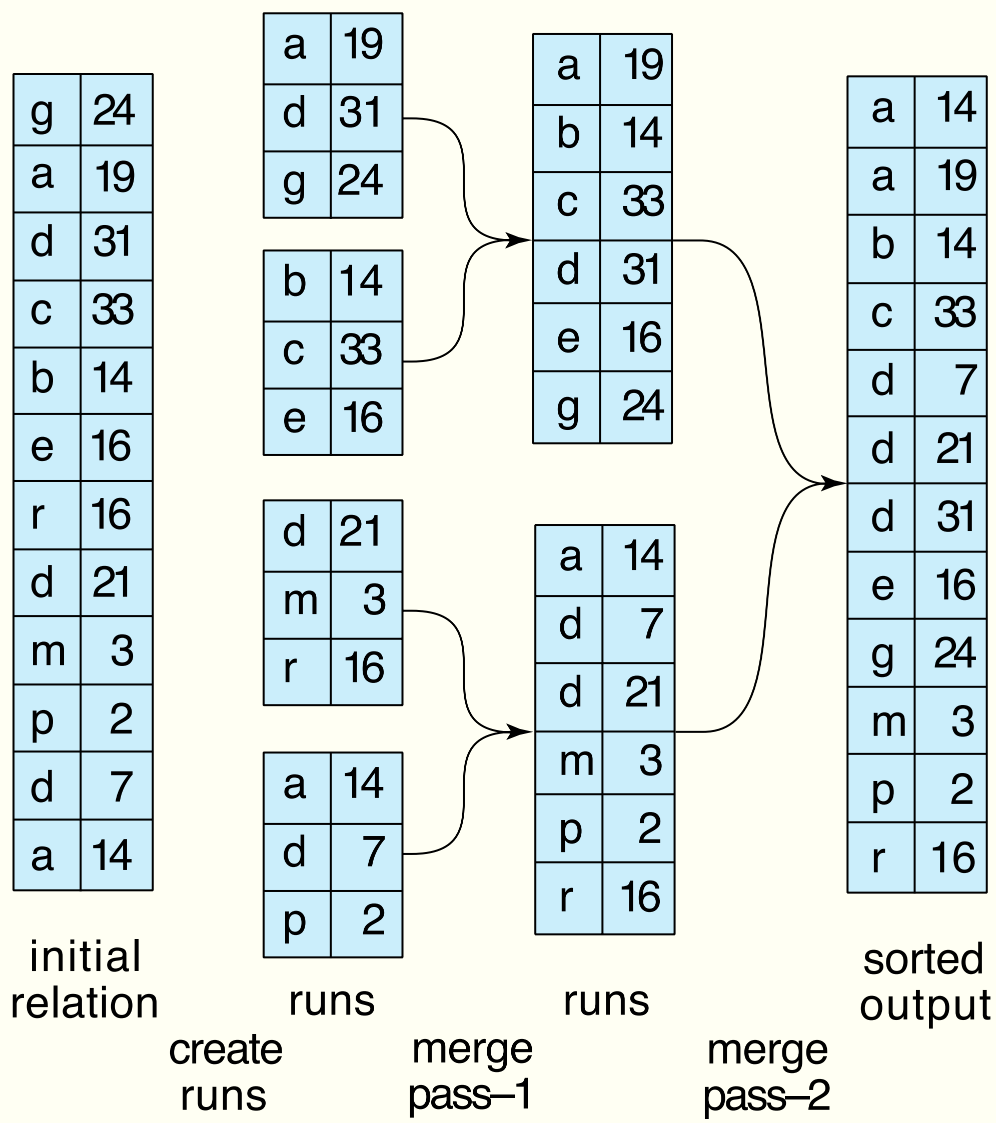
2. Merge the runs (N-way merge). Assume that N < M.
    1. Use N blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page
    2. repeat
        1. Select the first record (in sort order) among all buffer pages
        2. Write the record to the output buffer. If the output buffer is full write it to disk.
        3. Delete the record from its input buffer page.
           If the buffer page becomes empty then
              read the next block (if any) of the run into the b
    3. until all input buffer pages are empty:



External Merge Sort (Cont.)

- If $N >= M$, several merge passes are required.
  - In each pass, contiguous groups of $M - 1$ runs are merged.
  - A pass reduces the number of runs by a factor of $M - 1$, and creates runs longer by the same factor.
    - E.g. If M=11, and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
  - Repeated passes are performed till all runs have been merged into one.

Example: External Merge Sort



External Merge Sort Cost Analysis

- Cost analysis:
  - 1 block per run leads to too many seeks during merge
    - Instead use b_b buffer blocks per run
      ➔ read/write b_b blocks at a time
    - Can merge $\lfloor M/b_b \rfloor - 1$ runs in one pass
  - Total number of merge passes required: $\lceil \log_{\lfloor M/bb \rfloor - 1}(b_r/M) \rceil$.
    - b_r - number of blocks containing tuples of relation R
  - Block transfers for initial run creation as well as in each pass is $2b_r$
    - for final pass, we don't count write cost
      - we ignore final write cost for all operations since the output of an operation may be sent to the parent operation without being written to disk
    - Thus total number of block transfers for external sorting:

      $br\ (2 \lceil \log_{\lfloor M/bb \rfloor - 1} (b_r / M) \rceil + 1)$

Cost of Seeks

- During run generation: one seek to read each run and one seek to write each run
  - $2 \lceil br / M \rceil$
- During the merge phase
  - Need $2 \lceil br / bb \rceil$ seeks for each merge pass
    - except the final one which does not require a write
  - Total number of seeks:

$$2 \lceil br / M \rceil + \lceil br / bb \rceil (2 \lceil \log_{\lfloor M/bb \rfloor - 1}(br / M) \rceil - 1)$$