Lecture 20 Multikey Files

Last Day: Multikey Files

- Secondary Keys
- Inverted Lists

Today: Query Processing

- Ad hoc queries
- Query Language
- Parse Trees
- Answering queries
- Using indexes.

Multikey File Organization

- Direct Files and Indexed Files support efficient data access by a single Field (the key).
- eg, Given an index on Employee Number, the query "Retrieve employee #2317" is fast,
- But, "Retrieve all employees living in Toronto" is slow, since we must scan the entire file.
- Multi key files support fast access by several different fields,

Secondary Key Example

loc	E#	NAME	ADDRESS	AGE	SEX	SKILL
1	001	Hicks	Toronto	36	M	Programmer
2	020	McLeod	Montreal	51	M	Analyst
3	023	Lucas	Toronto	25	F	Technician
4	025	Bradley	Ottawa	35	F	Operator
5	030	Date	Montreal	45	M	Operator
6	045	Loomis	Vancouver	45	F	Analyst
7	046	Mader	Edmonton	38	M	Operator
8	048	Wu	Calgary	50	F	Programmer
9	055	Bair	Toronto	28	M	Analyst
10	060	Uhlig	Vancouver	24	M	Technician
11	062	Orilia	Montreal	21	M	Designer
12	070	Fry	Calgary	34	F	Operator
13	075	Riley	Ottawa	40	F	Designer

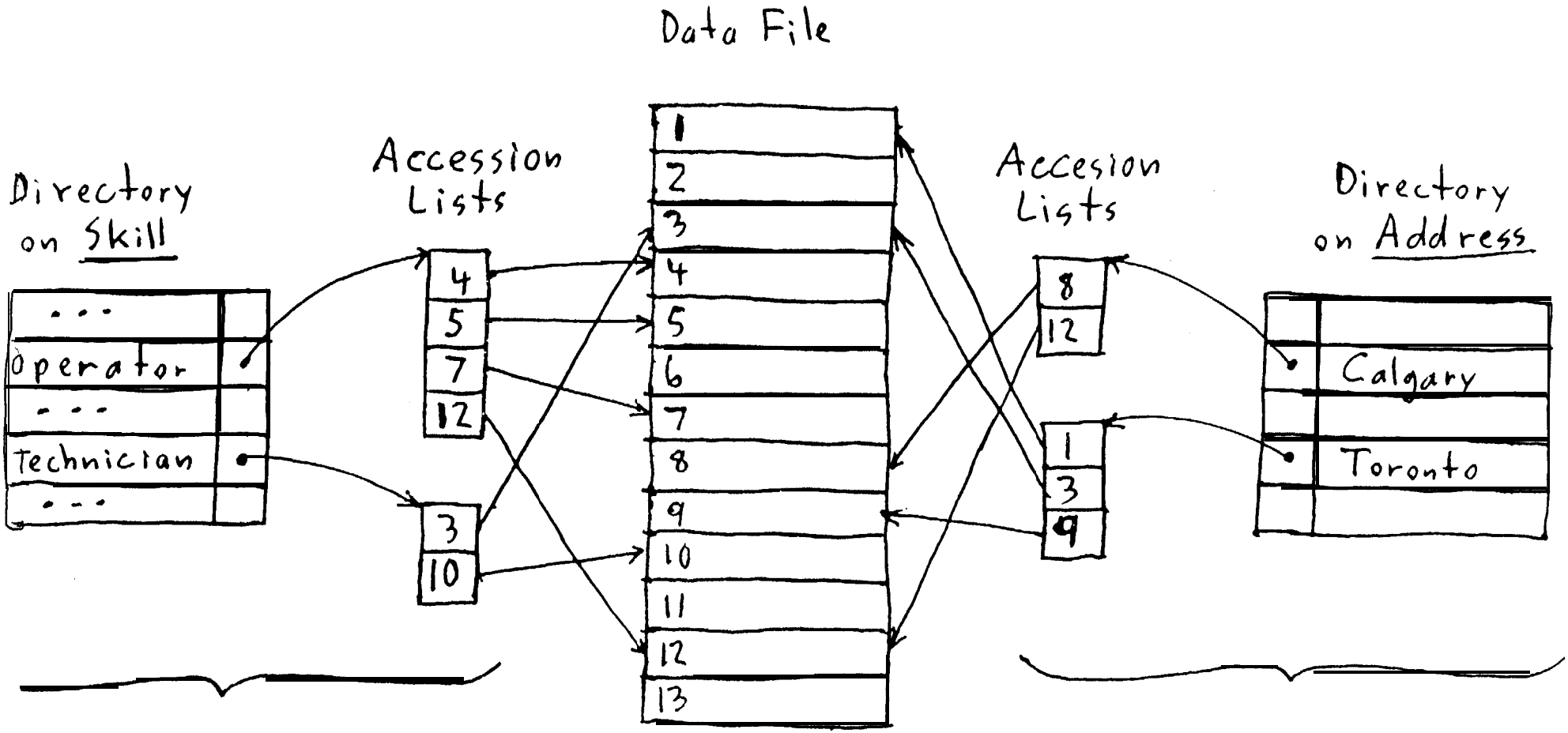
ADDRESS	loc
Calgary	8, 12
Edmonton	7
Montreal	2, 5, 11
Ottawa	9, 13
Toronto	1, 3, 9
Vancouver	6, 10

SKILL	loc
Analyst	2, 6, 9
Designer	11, 13
Operator	4, 5, 7, 12
Programmer	1, 8
Technician	3, 10

Key is E#

Secondary Keys are ADDRESS SKILL

Inverted Lists



Secondary key index on Skill.

Secondary key index on Address.

Ad Hoc Queries

- A user may request a selected set of records from a file, based on several fields

- eg, Retrieve the records of all female designers and programmers.

ie, Retrieve all records such that:

```
Sex = "F" AND (Skill = "Designer" OR
               Skill = "Programmer")
```

A query expressed in a formal language.

Query Processing

- The most efficient way to process a query depends on several factors:

  (1) The query

  (2) Secondary Indexes

  (3) The primary index

- There are several cases.

Case 1 : AND Queries

- eg, "Find all operators in Calgary"
- ie, Find all records such that

  Skill = "Operator" AND Address = "Calgary"

- First, Find the "Operator" entry in the Skill directory
- Then, get pointers in the accession list: {4, 5, 7, 12}
- Then, Find the "Calgary" entry in the Address directory
- Then, get pointers in the accession list: {8, 12}
- Then, intersect the two accession lists: {12}
- Finally, retrieve each record in the intersection set, ie Retrieve record 12.

Case 2: OR Queries

- eg, Retrieve all employees who are operators or who live in Calgary

- ie, Find all records such that

  Skill = "Operator" OR Address = "Calgary"

- Processing this query is the same as in the previous example, except now we union the two accession lists, to get {4, 5, 7, 8, 12}

- Then, we retrieve each record in this set from the data file.

Case 3: AND Queries, again.

- Life is not always so simple.

- eg. Retrieve all female programmers.
- ie, Retrieve all records such that

Sex = "F" AND Skill = "Programmer"

- This is an AND query, but there is no secondary index on Sex

- Method:
  - Find "Programmer" entry in Skill directory
  - Get pointers in accession list: {1, 8}
  - Retrieve the data records
  - Keep only those records with Sex = "F".

Case 4: OR Queries, again

- eg, Retrieve all employee records such that

  Sex = "F" OR Skill = "Programmer"

- Because there is no index on Sex, we must scan the entire file for female employees.

- So, the secondary index on Skill does not help at all here.

- Method:
  - Read each record in the file.
  - Keep those records with Sex = "f" or Skill = "Programmer".

Summary So Far

- 3 types of fields:
  - Fields with a secondary index (eg, Skill, Address
  - Fields with a primary index (eg, Emp#)
  - Fields with no index (eg, Age, Name, Sex)

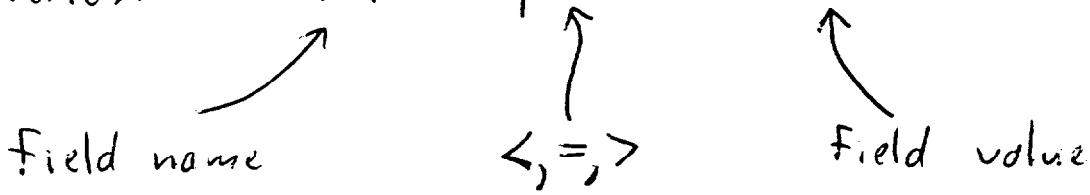
- 2 types of queries: And, Or

- ∴ 2×3 = 6 cases

  ie, 6 query processing methods.

Ad Hoc Query Language

- Basic Queries: Fn operator value



eg. Skill = "Technician"

Age > 25

- Boolean Queries:
  - Query 1 AND Query 2
  - Query 1 OR Query 2

eg. (Sex = "M" AND Age > 54) AND
(Skill = "Analyst" OR Skill = "Programmer")

Main Questions

① Given a file, some of whose fields are indexed, and given an arbitrary query, do the indexes help to answer the query?

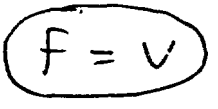
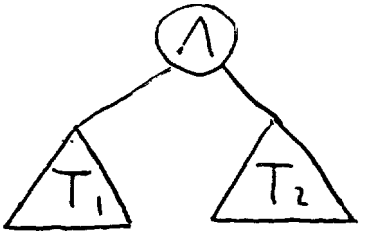
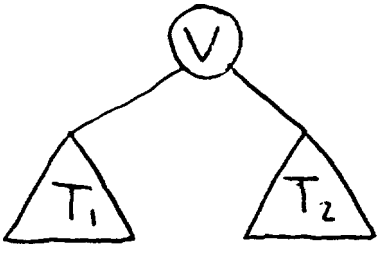
ie, do we need to scan the entire data file

② If the indexes do help, how should we use them to answer the query?

To answer these questions, first build the parse tree of the query.

Parse Trees

- A parse tree is a binary tree representing the "grammatical structure" of the query.
- They are defined recursively as follows:

Query	Parse Tree
f = v	
Query$_1$ AND Query$_2$	
Query$_1$ OR Query$_2$	

Here T_i is the parse tree for Query$_i$

Example

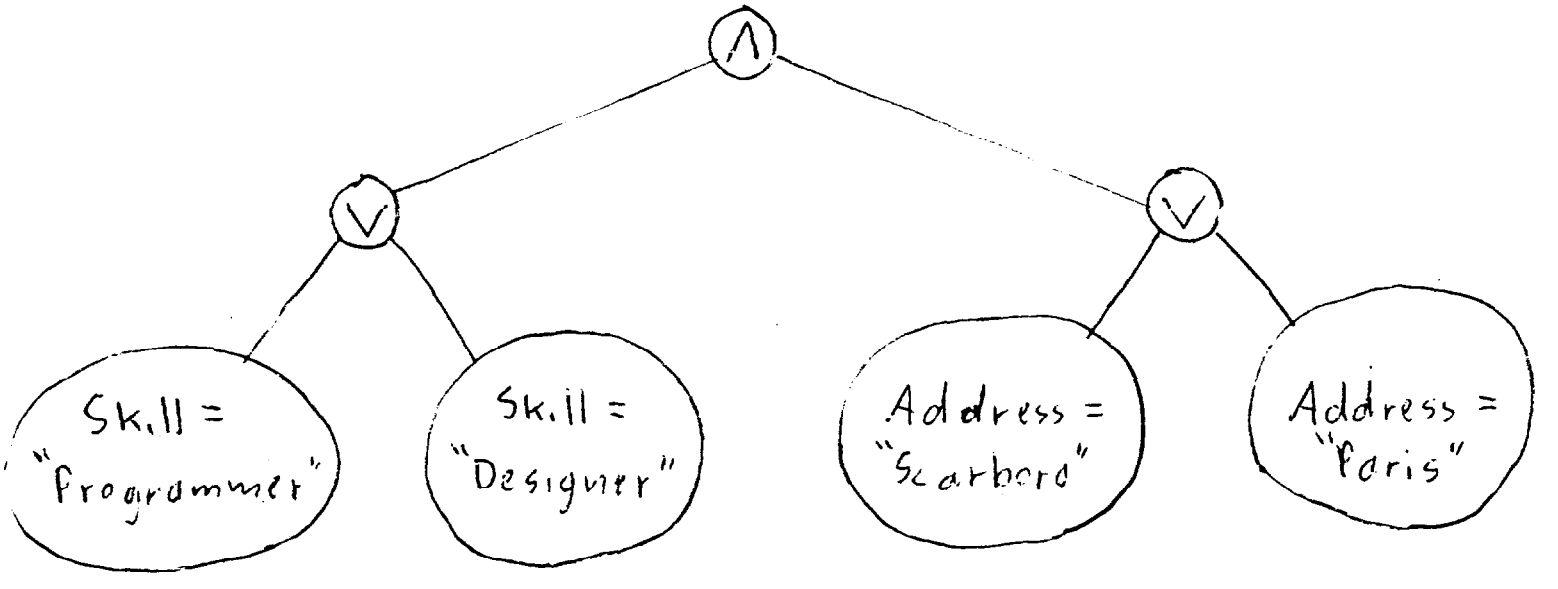
Query:

((Skill = "Programmer") OR (Skill = "Designer"

AND

((Address = "Scarboro") OR (Address = "Paris'

Its Parse Tree:

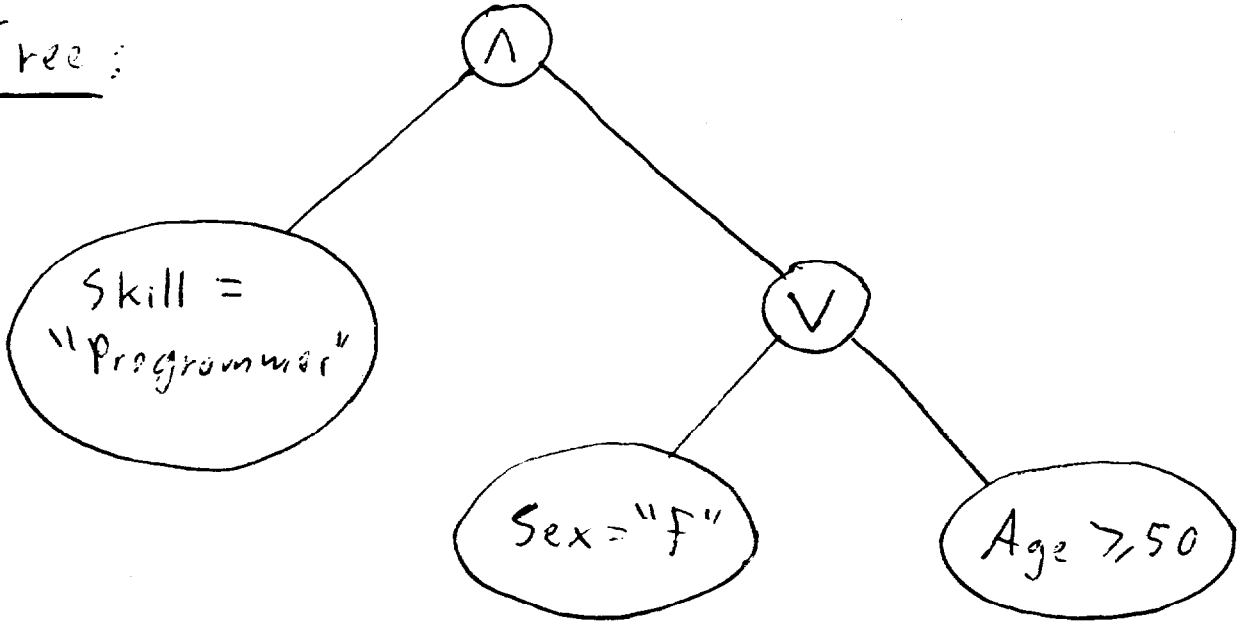


Given indexes on Address & Skill, do they help? (Yes, in this case.)

Do the Indexes Help: Example 1

Query: Skill = "Programmer"
AND (Sex = "F" OR Age ≥ 50)

Parse Tree:



- Indexes on Skill and Address.

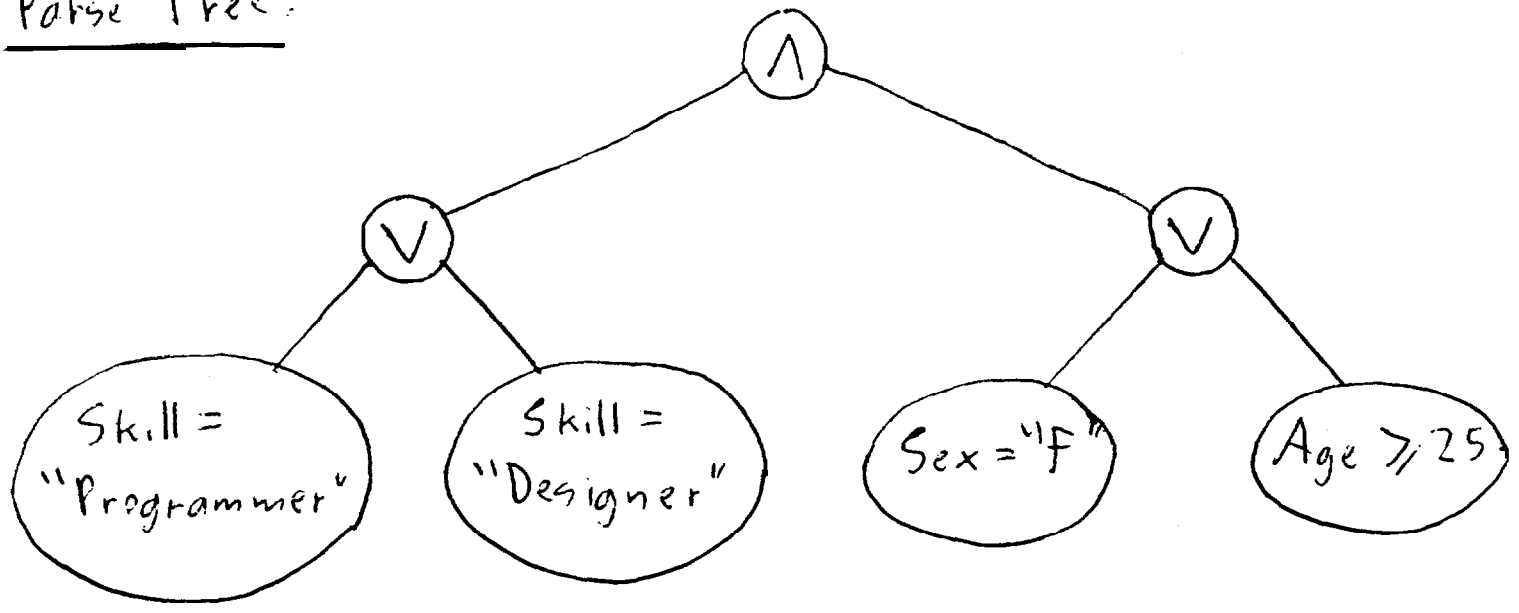
- In this case, they help

Do the Indexes Help: Example 2

Query:

(Skill = "Programmer") OR (Skill = "Designer")
AND
(Sex = "f") OR (Age ≥ 25)

Parse Tree:



- Indexes on Skill and Address.

- In this case, they help.

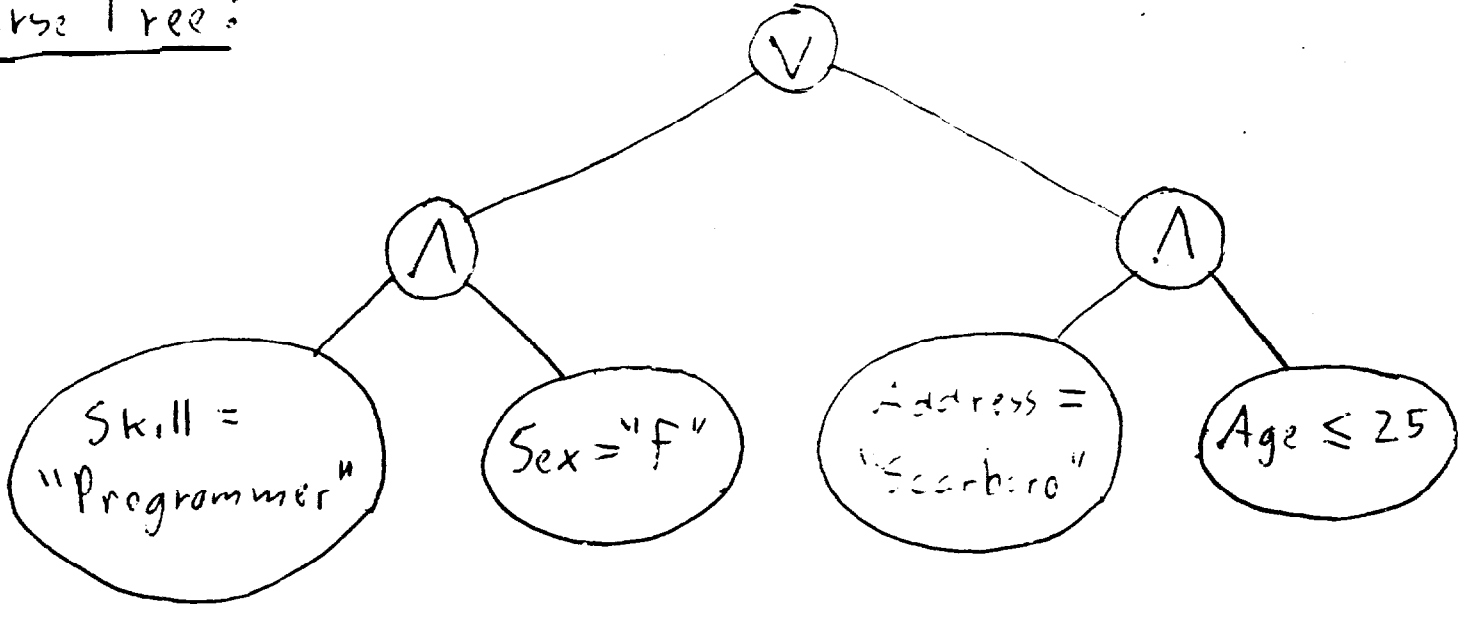
Do the Indexes Help : Example 3

Query: ((Skill = "Programmer") AND (Sex = "f"))

OR

((Address = "Scarboro") AND (Age ≤ 25))

Parse Tree:



- Indexes on Skill and Address

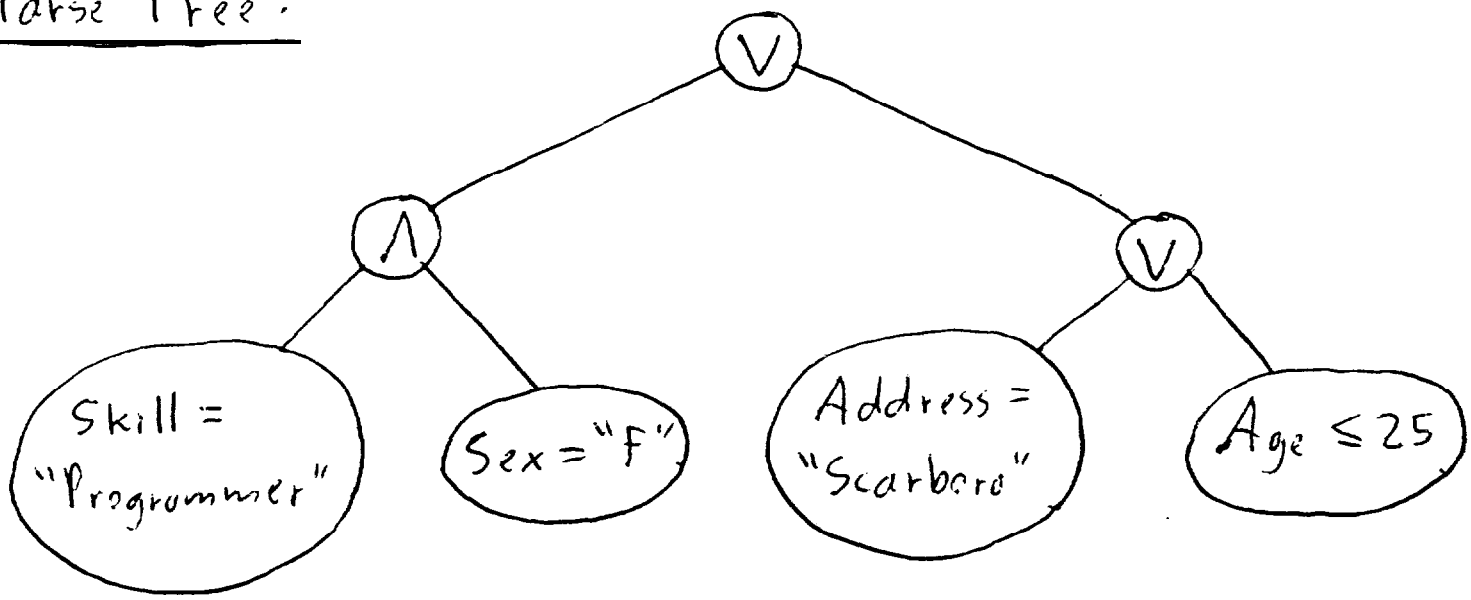
- In this case, they help.

Do the Indexes Help: Example 4

Query:

((Skill = "Programmer") AND (Sex = "F"))
OR
((Address = "Scarboro") OR (Age ≤ 25))

Parse Tree:



- Indexes on Skill and Address

- In this case, they do not help.

Do the Indices Help: In General

① If the query is of the form "F = v", then indices help if F is an indexed field.

② • If the query is of the form "Q_1 AND Q_2", then indices help if they help to answer <u>at least one</u> of Q_1, Q_2.

• If the query is of the form "Q_1 OR Q_2", then indices help if they help to answer <u>both</u> Q_1 & Q_2.

Algorithm 1

Do the indexes help to answer a query?

① Construct the parse tree of the query.

② Put a * next to each leaf that refers to an indexed field.

③ Loop:

- Choose a node, N.
- If N is an AND node with at least one *'ed child, then put a * next to N.
- If N is an OR node with both children *'ed, then put a * next to N.
- Exit when no new nodes can be *'ed.

end loop.

④ If the root is *'ed, then the indices help. Otherwise the entire file must be scanned

Parse Trees (Cont.)

- This starred parse tree is also useful for answering the second question:
  How to use the indices

- Some definitions:
  - If X is a node of a parse tree, then the query at X is the query represented by the subtree rooted at X.
  - For node X, let Xp denote the set of accession-list pointers to the records that satisfy the query at X.

Algorithm 2

Goal: Compute XP, assuming X is *'ed.

① If X is a leaf, then compute XP directly using an index.

② If X has the form "Y AND Z", then

Case 1: (Both Y and Z are *'ed)

Compute YP, ZP.

$XP := YP \cap ZP$

Case 2: (Only one of Y and Z is *'ed. Suppose Y)

Compute YP

$XP := \{\}$

for each $p \in YP$

Retrieve the record, r, pointed to by p.

If r satisfies the query at Z,

then $XP := XP \cup \{p\}$

Algorithm 2 (Cont.)

③ If X is of the form "Y OR Z", then both Y and Z must be *'ed.

Compute YP, ZP

XP := YP ∪ ZP

To Answer a Query

① Construct the parse tree of the query, Q.

② * the nodes using Algorithm I.

③ If the root is <u>not</u> *'ed, then scan the entire data file, and keep those records satisfying Q.

④ If the root, R, <u>is</u> *'ed, then compute RP, the pointers to the records in the query answer.

⑤ Retrieve these records from the data file.