Cost Model for Our Analysis

We ignore CPU costs, for simplicity:
- **B:** The number of data pages
- **R:** Number of records per page
- **D:** (Average) time to read or write disk page
- Measuring number of page I/O's ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

  ⊠ *Good enough to show the overall trends!*

Comparing File Organizations

- ❖ Heap files (random order; insert at eof)
- ❖ Sorted files, sorted on <*age, sal*>
- ❖ Clustered B+ tree file, Alternative (1), search key <*age, sal*>
- ❖ Heap file with unclustered B + tree index on search key <*age, sal*>
- ❖ Heap file with unclustered hash index on search key <*age, sal*>
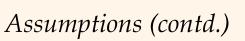
Operations to Compare

- ❖ Scan: Fetch all records from disk
- ❖ Equality search
- ❖ Range selection
- ❖ Insert a record
- ❖ Delete a record

Assumptions in Our Analysis

- ❖ Heap Files:
  - Equality selection on key; exactly one match.
- ❖ Sorted Files:
  - Files compacted after deletions.
- ❖ Indexes:
  - Alt (2), (3): data entry size = 10% size of record
  - Hash: No overflow buckets.
    - 80% page occupancy => File size = 1.25 data size
  - Tree: 67% occupancy (this is typical).
    - Implies file size = 1.5 data size

Assumptions (contd.)

- ❖ Scans:
  - ▪ Leaf levels of a tree-index are chained.
  - ▪ Index data-entries plus actual file scanned for unclustered indexes.
- ❖ Range searches:
  - ▪ We use tree indexes to restrict the set of data records fetched, but ignore hash indexes.

Cost of Operations

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap	BD	0.5BD	BD	2D	Search +D
(2) Sorted	BD	$D\log_2 B$	$D(\log_2 B +$ # pgs with match recs)	Search + BD	Search +BD
(3) Clustered	1.5BD	$D\log_F 1.5B$	$D(\log_F 1.5B$ + # pgs w. match recs)	Search + D	Search +D
(4) Unclust. Tree index	BD(R+0.15)	$D(1 + \log_F 0.15B)$	$D(\log_F 0.15B$ + # pgs w. match recs)	Search + 2D	Search + 2D
(5) Unclust. Hash index	BD(R+0.125)	2D	BD	Search + 2D	Search + 2D

✉ *Several assumptions underlie these (rough) estimates!*

Understanding the Workload

- ❖ For each query in the workload:
  - ▪ Which relations does it access?
  - ▪ Which attributes are retrieved?
  - ▪ Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- ❖ For each update in the workload:
  - ▪ Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
  - ▪ The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

Choice of Indexes

- ❖ What indexes should we create?
  - ▪ Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- ❖ For each index, what kind of an index should it be?
  - ▪ Clustered? Hash/tree?

Choice of Indexes (Contd.)

- ❖ One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
  - ▪ Obviously, this implies that we must understand how a DBMS evaluates queries and creates query evaluation plans!
  - ▪ For now, we discuss simple 1-table queries.
- ❖ Before creating an index, must also consider the impact on updates in the workload!
  - ▪ Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

Index Selection Guidelines

- ❖ Attributes in WHERE clause are candidates for index keys.
  - ▪ Exact match condition suggests hash index.
  - ▪ Range query suggests tree index.
    - • Clustering is especially useful for range queries; can also help on equality queries if there are many duplicates.
- ❖ Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
  - ▪ Order of attributes is important for range queries.
  - ▪ Such indexes can sometimes enable index-only strategies for important queries.
    - • For index-only strategies, clustering is not important!
- ❖ Try to choose indexes that benefit as many queries as possible. Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

Examples of Clustered Indexes

- ❖ B+ tree index on E.age can be used to get qualifying tuples.
  - ▪ How selective is the condition?
  - ▪ Is the index clustered?
- ❖ Consider the GROUP BY query.
  - ▪ If many tuples have *E.age* > 10, using *E.age* index and sorting the retrieved tuples may be costly.
  - ▪ Clustered *E.dno* index may be better!
- ❖ Equality queries and duplicates:
  - ▪ Clustering on *E.hobby* helps!

```
SELECT  E.dno
FROM  Emp E
WHERE  E.age>40
```

```
SELECT  E.dno, COUNT (*)
FROM  Emp E
WHERE  E.age>10
GROUP BY E.dno
```

```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby=Stamps
```

Index-Only Plans

- ❖ A number of queries can be answered without retrieving any tuples from one or more of the relations involved if a suitable index is available.

<E.dno>
```
SELECT  E.dno, COUNT(*)
FROM  Emp E
GROUP BY E.dno
```

<E.dno,E.sal>
Tree index!
```
SELECT  E.dno, MIN(E.sal)
FROM  Emp E
GROUP BY  E.dno
```

<E. age,E.sal>
or
<E.sal, E.age>
Tree index!
```
SELECT AVG(E.sal)
FROM  Emp E
WHERE E.age=25 AND
   E.sal BETWEEN 3000 AND 5000
```

Index-Only Plans (Contd.)

❖ Index-only plans
  are possible if the
  key is <dno,age>
  or we have a tree
  index with key
  <age,dno>
  ▪ Which is better?
  ▪ What if we
    consider the
    second query?

```
SELECT  E.dno,  COUNT (*)
FROM  Emp E
WHERE  E.age=30
GROUP BY E.dno
```

```
SELECT  E.dno,  COUNT (*)
FROM  Emp E
WHERE  E.age>30
GROUP BY E.dno
```

Index-Only Plans (Contd.)

❖ Index-only
  plans can also
  be found for
  queries
  involving more
  than one table;
  more on this
  later.

<E.dno>

```
SELECT  D.mgr
FROM  Dept D, Emp E
WHERE  D.dno=E.dno
```

<E.dno,E.eid>

```
SELECT  D.mgr, E.eid
FROM  Dept D, Emp E
WHERE  D.dno=E.dno
```