Cost Models

Operations on an Index

- Search: Given a key find all records
  - More sophisticated variants as well. Why?

- Insert /Remove entries
  - Bulk Load. Why?

Real difference between structures:
costs of operations

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					
(2) Sorted					
(3) Clustered					
(4) Unclustered Tree index					
(5) Unclustered Hash index					

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

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!*

Putting the cost to use: Choosing Indexes

Overview of where we are

- We've seen pages, records, files, and indexes
- We know what operations they perform and that cost is really important
- Next:
  - (1) How indexes used to optimize simple queries
  - (2) Nitty gritty details of indexing (the algorithms)

Tuning

- Fundamental tradeoff: Each index speeds up some operations, but slows down others

- How do we choose? Examine our *workload*:
  - set of queries and updates we run against the db,
  - and how frequently we run each one.

In this lecture, Tuning maps the workload into a set of choices: Create a clustered hash table on Employee.Name.

Questions about 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?

Key questions to ask for each query in your workload

Understanding the updates

- 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 index?
  - Clustered? Hash/tree?

Choice of Indexes

- One approach: Consider the most important queries in turn.

- Key Question: Adding an index improve the plan? Yes, create it. Does it hurt update rates?

To fully answer need to understand query evaluation in detail (much later).
1 table queries for now.

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

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

Index on age makes sense, what kind?

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

What kind of index?

How would you choose to cluster?

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

Composite Keys

11	80
12	10
12	20
13	75

<Age, Sal>

Name	Age	Sal
Bob	12	10
Cal	11	80
Joe	12	20
Sue	13	75

Equality Query:
Age = 12 and sal = 90?

Range Query:
Age = 5 and sal > 5?

80	11
10	12
20	12
75	13

<Sal, Age>

11
12
12
13

<Age>

80
10
20
75

<Sal>

Composite keys in *Dictionary Order*

Composite keys

- Pro:
  - when they work they work well
  - We'll see a good case called "index-only" plans
- Con:
  - Guesses? (time and space)

Index-Only Plans

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

What kind of index to make each query index-only?

```
SELECT  E.dno, MIN(E.sal)
FROM  Emp E
GROUP BY  E.dno
```

```
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 possible when

1. the key is <dno,age>
2. we have a tree index with key <age,dno>

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

Wait -- we use constraints to optimize? This is probably a shock

Index-Only Plans (Contd.)

- Index-only plans for queries involving more than one table; more 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
```

Summary

- Many alternative file organizations exist, each appropriate in some situation.
- Index is a collection of data entries plus a way to quickly find entries with given key values.
- If selection queries are frequent, sorting the file, or building an *index* is important.
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)

Summary (Contd.)

- Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
  - Choice orthogonal to *indexing technique* used to locate data entries with a given key value.
- Can have several indexes on a given file of data records, each with a different search key.
- Indexes can be classified as clustered vs. unclustered, primary vs. secondary, and dense vs. sparse. Differences have important consequences for utility/performance.

Summary (Contd.)

- Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?
- Indexes must be chosen to speed up important queries (and perhaps some updates!).
  - Index maintenance overhead on updates to key fields.
  - Choose indexes that can help many queries, if possible.
  - Build indexes to support index-only strategies.
  - Clustering is an important decision; only one index on a given relation can be clustered!
  - Order of fields in composite index key can be important.