Intro to Index (on files)

- Suppose we have a relation $R$
  $\rightarrow$ stored in a number of blocks on disk.

- User query:

  \[
  \text{select * from } R \text{ where } a = 10
  \]

- Without further info., the only way to process this query is:

  1. read all blocks of the relation $R$
  2. look for tuples with attr. value $a = 10$

- Index:

  additional info. on specific attribute(s) to speed up lookup of tuples.
Definition:

Search key = field(s) used to create the index.

Index file = file that store records of the form:

[search-key] pointer

(read pointer into a data file or bucket file)

(NB: size (index file) << size of (data file)).

(several orders)

2 Basic kinds of indices:

(1) Ordered indices

(2) Hash indices

(Hash Tables)
Operation/Effect of an Index:

Search key value → Index → All blocks holding records → Records that contain matching search key value.
Index - Structure Basics

Data File:

Usually: Primary Key
& (Sort Key).

<table>
<thead>
<tr>
<th>10</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>30</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>50</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>70</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>90</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>F</td>
</tr>
</tbody>
</table>

Sequential File
* Sequential file = a file containing records that are sorted by some attribute(s) that usually the primary key.

Tuples (records) are stored in the sorted order!!!

(NB: common practice in sequential files)

1. Leave room in a block for future tuples.
2. Use overflow blocks.

* Sort key = field(s) whose values are used to sort/order tuples in a sequential file.
Ordered Index = an index structure where the index entries:

| search_key | pointer |

are stored SORTED (on the search_key) in an index file.

CNB: take "sorted" with a grain of salt. B-tree is an ordered index. B-tree store the index-entries non-sequentially but still, there is an order (sorted order!).

Primary Index = is an ordered index.

The search key in the index entries:

| search key | pointer |

of a primary index, is ALSO the sort key used for the sequential file.
Example:

Primary Index (file):

Primary Index:

Data File

Secondary Index:

The search key in the index entries

If a secondary index, is

NOT a sort key
Index-sequential file =

a sequential file that

has a primary index

(made popular by IBM).
Dense Index = The index file contains an index entry for every value of the search key in the data file.

* Note: a secondary index is always dense.
- **Sparse index** = The index file's index entry does not contain every value of the search key in the data file.

  Example:

  \[
  \begin{array}{c|c}
  \text{Sparse index} & \text{Search-key} \mid \text{Pointer} \\
  \hline
  10 & 10 \\
  30 & 20 \\
  50 & 30 \\
  70 & 40 \\
  90 & 50 \\
  \end{array}
  \]

  Each index entry points to a range of values, reducing space and gaining time!!

- **NOTE**: Only the primary index can be sparse!!!

  **NOTE**: Generally slower than dense indexes due to writing needs.
Multi-level Index.

- Fact: An ordered index is a file
- Example: We can put a (primary) index on this file (to speed up searching for a search key!!!)

[Diagram of index and secondary index]

Find (D)
Note: Later, we will study a dynamic multi-level index data structure.

⇒ B-tree.

very popular (most popular)

ordered index structure!!!
Indirection in secondary indexes.

- "Normal" secondary index:

  Data File

  "Normal" secondary index

  10
  20

  A
  B

  30
  40

  B
  A

  50
  60

  C
  B

  70
  80

  B
  E

  90
  100

  C
  F

A search key that appears \( n \) times in the Data file will appear \( n \) times in the index file.
Secondary index with "Indirection": 

You should use this ONLY when the search key is LARGE!!
Similar Application: Document retrieval.

- Q: How to find documents (quickly) that contain certain key words.

- Visualize a document as a relation:
  \[
  \text{Doc ( hasCat, hasDog, ... )} \]

Example:

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>doc2</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>doc3</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Key: we DO NOT index tuples that contains the \text{FALSE} value.
First iteration: one index file for each attr.

Documents

Cat

Dog

A
B
C
D

TF
FF
FT

Better: combine indices with a second level.

Buckets

Cat

Dog

A
B
C
D

Documents

TT
TF
FF
FT

This is called an "inverted index".
More informative bucket file

- When indexing HTML/XML documents, the occurrence of the word has a context associated with it.

- This context (tag) can be stored in the bucket file:

```
<table>
<thead>
<tr>
<th>inverted index</th>
<th>type</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>header</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>57</td>
</tr>
</tbody>
</table>
```

Refer to document.