Lecture 17: Data Storage

Friday, November 8, 2006

Outline

- Representing data elements (chapter 12)
- Index structures (13.1, 13.2)

Representing Data Elements

- Relational database elements:

```sql
CREATE TABLE Product (
    pid INT PRIMARY KEY,
    name CHAR(20),
    description VARCHAR(200),
    maker CHAR(10) REFERENCES Company(name)
)
```

- A tuple is represented as a record

Record Formats: Fixed Length

```
F1   F2   F3   F4
L1   L2   L3   L4
```

Base address (B)  Address = B+L1+L2

- Information about field types same for all records in a file; stored in system catalogs.
- Finding i'th field requires scan of record.
- Note the importance of schema information!

Record Header

```
To schema
length
F1   F2   F3   F4
L1   L2   L3   L4
```

Need the header because:
- The schema may change for a while new/old may coexist
- Records from different relations may coexist

Variable Length Records

```
Other header information
F1   F2   F3   F4
L1   L2   L3   L4
```

Place the fixed fields first: F1, F2
Then the variable length fields: F3, F4
Null values take 2 bytes only
Sometimes they take 0 bytes (when at the end)
**Records With Repeating Fields**

Other header information

- L1
- L2
- L3

Needed e.g. in Object Relational systems, or fancy representations of many-many relationships

---

**Storing Records in Blocks**

- Blocks have fixed size (typically 4k)

---

**Spanning Records Across Blocks**

- When records are very large
- Or even medium size: saves space in blocks

---

**BLOB**

- Binary large objects
- Supported by modern database systems
- E.g. images, sounds, etc.
- Storage: attempt to cluster blocks together

**CLOB = character large objec**

- Supports only restricted operations

---

**Modifications: Insertion**

- File is unsorted: add it to the end (easy ☺)
- File is sorted:
  - Is there space in the right block?
    - Yes: we are lucky, store it there
  - Is there space in a neighboring block?
    - Look 1-2 blocks to the left/right, shift records
  - If anything else fails, create overflow block

---

**Overflow Blocks**

- After a while the file starts being dominated by overflow blocks: time to reorganize
Modifications: Deletions

- Free space in block, shift records
- Maybe be able to eliminate an overflow block
- Can never really eliminate the record, because others may point to it
  - Place a tombstone instead (a NULL record)

Modifications: Updates

- If new record is shorter than previous, easy 🙌
- If it is longer, need to shift records, create overflow blocks

Physical Addresses

- Each block and each record have a physical address that consists of:
  - The host
  - The disk
  - The cylinder number
  - The track number
  - The block within the track
  - For records: an offset in the block
    - sometimes this is in the block’s header

Logical Addresses

- Logical address: a string of bytes (10-16)
- More flexible: can blocks/records around
- But need translation table:

<table>
<thead>
<tr>
<th>Logical address</th>
<th>Physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>P1</td>
</tr>
<tr>
<td>L2</td>
<td>P2</td>
</tr>
<tr>
<td>L3</td>
<td>P3</td>
</tr>
</tbody>
</table>

Main Memory Address

- When the block is read in main memory, it receives a main memory address
- Need another translation table

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Logical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>L1</td>
</tr>
<tr>
<td>M2</td>
<td>L2</td>
</tr>
<tr>
<td>M3</td>
<td>L3</td>
</tr>
</tbody>
</table>

Optimization: Pointer Swizzling

- ≠ the process of replacing a physical/logical pointer with a main memory pointer
- Still need translation table, but subsequent references are faster
**Pointer Swizzling**

- **Automatic**: when block is read in main memory, swizzle all pointers in the block
- **On demand**: swizzle only when user requests
- **No swizzling**: always use translation table

---

**Index Classification**

- Primary/secondary
- Clustered/unclustered
- Dense/sparse
- B+ tree / Hash table / …

---

**Indexes**

- An index on a file speeds up selections on the search key fields for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - Search key is not the same as key (minimal set of fields that uniquely identify a record in a relation).
- An index contains a collection of data entries, and supports efficient retrieval of all data entries with a given key value k.

---

**Primary Index**

- File is sorted on the index attribute
- Dense index: sequence of (key, pointer) pairs
Primary Index

- **Sparse** index

Primary Index with Duplicate Keys

- Sparse index: pointer to lowest search key in each block:

- Search for 20

Primary Index with Duplicate Keys

- Better: pointer to \textit{lowest new search key} in each block:

- Search for 20

- Search for 15 ? 35 ?

Secondary Indexes

- To index other attributes than primary key
- Always dense (why?)

Clustered/Unclustered

- Primary indexes = usually clustered
- Secondary indexes = usually unclustered
Clustered vs. Unclustered Index

Secondary Indexes

- Applications:
  - index other attributes than primary key
  - index unsorted files (heap files)
  - index clustered data

Applications of Secondary Indexes

- **Clustered data**

  Company(name, city), Product(pid, maker)

  Select city
  From Company, Product
  Where name=maker
  and pid="p045"

  Select pid
  From Company, Product
  Where name=maker
  and city="Seattle"

Composite Search Keys

- **Composite Search Keys**: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
    - age=20 and sal=75
  - Range query: Some field value is not a constant. E.g.:
    - age ~20; or age~20 and sal > 10

Examples of composite key indexes using lexicographic order:

<table>
<thead>
<tr>
<th>Company 1</th>
<th>Company 2</th>
<th>Company 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Products of company 1</th>
<th>Products of company 2</th>
<th>Products of company 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company 1</td>
<td>Company 2</td>
<td>Company 3</td>
</tr>
<tr>
<td>Data entries</td>
<td>Data entries</td>
<td>Data entries</td>
</tr>
<tr>
<td>sorted by &lt;sal,age&gt;</td>
<td>sorted by &lt;sal,age&gt;</td>
<td>sorted by &lt;sal,age&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data entries in index</th>
<th>Data entries in index</th>
</tr>
</thead>
<tbody>
<tr>
<td>sorted by &lt;sal,age&gt;</td>
<td>sorted by &lt;sal,age&gt;</td>
</tr>
</tbody>
</table>