Outline

- It’s all about disks!
  - That's why we always draw databases as 📁
  - And why the single most important metric in database processing is the number of disk I/O's performed

- Record layout
- Block layout

How far away is data?

<table>
<thead>
<tr>
<th>Location</th>
<th>Time</th>
<th>Location</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>1 min.</td>
<td>My head</td>
<td>1</td>
</tr>
<tr>
<td>On-chip cache</td>
<td>2 min.</td>
<td>This room</td>
<td>2</td>
</tr>
<tr>
<td>On-board cache</td>
<td>10 min.</td>
<td>Duke campus</td>
<td>10</td>
</tr>
<tr>
<td>Memory</td>
<td>100cycles.</td>
<td>Washington D.C.</td>
<td>1.5 hr.</td>
</tr>
<tr>
<td>Disk</td>
<td>$10^6$</td>
<td>Pluto</td>
<td>2 yr.</td>
</tr>
<tr>
<td>Tape</td>
<td>$10^6$</td>
<td>Andromeda</td>
<td>2000 yr.</td>
</tr>
</tbody>
</table>

(Source: AlphaSort paper, 1995)

I/O dominates—design your algorithms to reduce I/O!

A typical disk

- Spindle rotation
- Arm movement
- "Moving parts" are slow

Top view

Higher-density sectors on inner tracks and/or more sectors on outer tracks

A block is a logical unit of transfer consisting of one or more sectors
Disk access time

Sum of:
- Seek time: time for disk heads to move to the correct cylinder
- Rotational delay: time for the desired block to rotate under the disk head
- Transfer time: time to read/write data in the block (= time for disk to rotate over the block)

Random disk access

Seek time + rotational delay + transfer time

- Average seek time
  - Time to skip one half of the cylinders?
  - Not quite; should be time to skip a third of them (why?)
  - “Typical” value: 5 ms
- Average rotational delay
  - Time for a half rotation (a function of RPM)
  - “Typical” value: 4.2 ms (7200 RPM)

Sequential disk access

Seek time + rotational delay + transfer time

- Seek time
  - 0 (assuming data is on the same track)
- Rotational delay
  - 0 (assuming data is in the next block on the track)
- Transfer time: time to read/write data in the block (= time for disk to rotate over the block)

Data layout strategy

Keep related things close together!
- Same sector/block
- Same track
- Same cylinder
- Adjacent cylinder

More performance tricks

- Disk scheduling algorithm
  - Example: “elevator” algorithm
- Track buffer
  - Read/write one entire track at a time
- Double buffering
  - While processing the current block in memory, prefetch the next block from disk
- Parallel I/O
  - More disk heads working at the same time

Record layout

Record = row in a table

- Variable-format records
  - Rare in DBMS—table schema dictates the format
  - Relevant for semi-structured data such as XML
- Focus on fixed-format records
  - With fixed-length fields only, or
  - With possible variable-length fields
Fixed-length fields

- All field lengths and offsets are constant
  - Computed from schema, stored in the system catalog
- Example: `CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT);`

<table>
<thead>
<tr>
<th>Field</th>
<th>Start</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>name</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>age</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>GPA</td>
<td>56</td>
<td>36</td>
</tr>
</tbody>
</table>

- Bart (padded with \0)

- Watch out for alignment
  - May need to pad; reorder columns if that helps
- What about NULL?
  - Add a bitmap at the beginning of the record

Variable-length records

- Example: `CREATE TABLE Student(SID INT, name VARCHAR(20), age INT, GPA FLOAT, comment VARCHAR(100));`

- Approach 1: use field delimiters
  
<table>
<thead>
<tr>
<th>Field</th>
<th>Start</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>name</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>age</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>GPA</td>
<td>24</td>
<td>32</td>
</tr>
</tbody>
</table>

  0 4 8 16
  142 10 2.3 Bart\0 Weird kid\0

- Approach 2: use an offset array

<table>
<thead>
<tr>
<th>Field</th>
<th>Start</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>name</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>age</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>GPA</td>
<td>24</td>
<td>32</td>
</tr>
</tbody>
</table>

  0 4 8 16 24 32
  142 10 2.3 Bart Weird kid

- Put all variable-length fields at the end (why?)
- Update is messy if it changes the length of a field

LOB fields

- Example: `CREATE TABLE Student(SID INT, name CHAR(20), age INT, GPA FLOAT, picture BLOB(32000));`

- Student records get “de-clustered”
  - Bad because most queries do not involve picture
- Decomposition (automatically done by DBMS and transparent to the user)
  - `Student(SID, name, age, GPA)`
  - `StudentPicture(SID, picture)`

Block layout

How do you organize records in a block?

- NSM (N-ary Storage Model)
  - Most commercial DBMS
- PAX (Partition Attributes Across)
  - Recent work (Ailamaki et al., VLDB 2001)

Options

- Reorganize after every update/delete to avoid fragmentation (gaps between records)
  - Need to rewrite half of the block on average
- What if records are fixed-length?
  - Reorganize after delete
    - Only need to move one record
    - Need a pointer to the beginning of free space
  - Do not reorganize after update
    - Need a bitmap indicating which slots are in use
Cache behavior of NSM

- Query: SELECT SID FROM Student WHERE GPA > 2.0;
- Assumption: cache block size < record size
- Lots of cache misses
  - ID and GPA are not close enough by memory standards

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>3.1</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>4.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>3.1</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>2.3</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>4.3</td>
</tr>
</tbody>
</table>

PAX

- Most queries only access a few columns
- Cluster values of the same columns in each block
  - When a particular column of a row is brought into the cache, the same column of the next row is brought in together

Reorganize after every update (for variable-length records only) and delete to keep fields together

Summary

- Storage hierarchy
  - Why I/O's dominate the cost of database operations
- Disk
  - Steps in completing a disk access
  - Sequential versus random accesses
- Record layout
  - Handling variable-length fields
  - Handling NULL
  - Handling modifications
- Block layout
  - NSM: the traditional layout
  - PAX: a layout that tries to improve cache performance