Outline

- Storage hierarchy
- Disks
- RAID
- Buffer Manager
- File Organization
- Indexes

Redundant Array Independent Disks

RAID 0 – Blocks striped. No mirror. No parity.

Fast!

RAID 1 – Blocks mirrored. No stripe. No parity.

Redundant!

RAID 2 – Blocks striped. (and stores ECC)

Weird!

RAID 5 – Blocks striped. Distributed Parity.
RAID Level 5

- Distributed parity "blocks" instead of bits
- Normal operation:
  - "Read" directly from single disk.
  - Load distributed across all 5 disks
  - "Write": Need to read and update the parity block
    - To update 9 to 9'
      - read 9 and P2
      - compute P2' = P2 \text{ xor } 9 \text{ xor } 9'
      - write 9' and P2'

<table>
<thead>
<tr>
<th>P0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>P1</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>P2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>P3</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>P4</td>
</tr>
</tbody>
</table>

(raid5: block-interleaved distributed parity)

RAID Level 5

- Failure operation (disk 3 has failed)
  - "Read block 0": Read it directly from disk 2
  - "Read block 1" (which is on disk 3)
    - Read P0, 0, 2, 3 and compute 1 = P0 \text{ xor } 0 \text{ xor } 2 \text{ xor } 3
  - "Write"
    - To update 9 to 9'
      - read 9 and P2
      - Oh... P2 is on disk 3
      - So no need to update it
      - Write 9'

<table>
<thead>
<tr>
<th>P0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>P1</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>P3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>19</td>
<td>P4</td>
<td></td>
</tr>
</tbody>
</table>

(raid5: block-interleaved distributed parity)
Choosing a RAID level

- RAID 0 striping fastest, but no fault tolerance
- Main choice between RAID 1 and RAID 5
- Level 1 better write performance than level 5
  - Level 5: 2 block reads and 2 block writes to write a single block
  - Level 1: only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 5 lower storage cost
  - Usable storage for Level 1 only 50% of raw disk capacity
  - Level 5 is preferred for applications with low update rate, and large amounts of data

Query Processing/Storage

- Given a query, decide how to “execute” it
- Specify sequence of pages to be brought in memory
- Operate upon the tuples to produce results

- Bringing pages from disk to memory
- Managing the limited memory

- Storage hierarchy
- How are relations mapped to files?
- How are tuples mapped to disk blocks?
Buffer Manager

- When the QP wants a block, it asks the “buffer manager”
  - The block must be in memory to operate upon
- Buffer manager:
  - If block already in memory: return a pointer to it
  - If not:
    - Evict a current page
      - Either write it to temporary storage,
      - or write it back to its original location,
      - or just throw it away (if it was read from disk, and not modified)
  - and make a request to the storage subsystem to fetch it

Buffer Manager

Page Requests from Higher Levels

BUFFER POOL

disk page

free frame

MAIN MEMORY

DISK

DB

choice of frame dictated by replacement policy
Buffer Manager

- Similar to virtual memory manager
- Buffer replacement policies
  - What page to evict?
  - LRU: Least Recently Used
    - Throw out the page that was not used in a long time
  - MRU: Most Recently Used
    - The opposite
    - Why?
  - Clock?
    - An efficient implementation of LRU

Buffer Manager Requirements

- Pinning a block
  - Not allowed to evict
- Force-output (force-write)
  - Force the contents of a block to be written to disk
- Order the writes
  - This block must be written to disk before this block

Critical for fault tolerant guarantees
- Otherwise database has no control over what is on disk
Outline

- Storage hierarchy
- Disks
- RAID
- Buffer Manager
- File Organization
- Etc....

File Organization

- How are the relations mapped to the disk blocks?
  - Use a standard file system?
    - High-end systems have their own OS/file systems
    - OS interferes more than helps in many cases
  - Mapping of relations to file?
    - One-to-one?
    - Advantages in storing multiple relations clustered together
  - A file is essentially a collection of disk blocks
    - How are the tuples mapped to the disk blocks?
    - How are they stored within each block
File Organization

- Goals:
  - Allow insertion/deletions of tuples/records
  - Fetch a particular record (specified by record id)
  - Find all tuples that match a condition (say SSN = 123)

- Simplest case
  - Each relation is mapped to a file
  - A file contains a sequence of records
  - Each record corresponds to a logical tuple

- Next:
  - How are tuples/records stored within a block?

Fixed Length Records

- \( n = \) number of bytes per record
- Store record \( i \) at position:
  - \( n \times (i-1) \)

- Records may cross blocks
  - Not desirable
  - Stagger so that that doesn't happen

- Inserting a tuple?
  - Depends on the policy used
  - One option: Simply append at the end of the file. Problems?

- Deletions?
  - Option 1: Rearrange
  - Option 2: Keep a free list and use for next insert

The above assumes records not ordered.
Fixed Length Records

- Deleting: using “free lists”

<table>
<thead>
<tr>
<th>header</th>
<th>record 0</th>
<th>10101</th>
<th>Srinivasan</th>
<th>Comp. Sci.</th>
<th>65000</th>
</tr>
</thead>
<tbody>
<tr>
<td>record 1</td>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td></td>
</tr>
<tr>
<td>record 2</td>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td></td>
</tr>
<tr>
<td>record 3</td>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
<td></td>
</tr>
<tr>
<td>record 4</td>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
<td></td>
</tr>
<tr>
<td>record 5</td>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
<td></td>
</tr>
<tr>
<td>record 6</td>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
<td></td>
</tr>
<tr>
<td>record 7</td>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td></td>
</tr>
<tr>
<td>record 8</td>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
<td></td>
</tr>
</tbody>
</table>

Variable-length Records

Slotted page structure

- Indirection:
  - The records may move inside the page, but the outside world is oblivious to it
  - Why?
    - The headers are used as an indirection mechanism
    - Record ID 1000 is the 5th entry in the page number X
File Organization

- Which block of a file should a record go to?
  - Anywhere?
    - How to search for “SSN = 123”? 
    - Called “heap” organization
  - Sorted by SSN?
    - Called “sequential” organization
    - Keeping it sorted would be painful
    - How would you search?
  - Based on a “hash” key
    - Called “hashing” organization
    - Store the record with SSN = x in the block number x%1000
    - Why?

Sequential File Organization

- Keep sorted by some search key
- Insertion
  - Find the block in which the tuple should be
  - If there is free space, insert it
  - Otherwise, must create overflow pages
- Deletions
  - Delete and keep the free space
  - Databases tend to be insert heavy, so free space gets used fast
- Can become fragmented
  - Must reorganize once in a while
Sequential File Organization

- What if I want to find a particular record by value?
  - Account info for SSN = 123
- Binary search
  - Takes \(\text{ceiling}(\log_2(n))\) number of disk accesses
  - Random accesses
- Too much
  - \(n = 1,000,000,000 - \log_2(n) = 30\)
  - Recall each random access approx 10 ms
  - 300 ms to find just one account information
  - < 4 requests satisfied per second