Topics

- Disks
  - Hardware
  - Modeling Access Time
  - Disk Scheduling

- Buffer Management
Disk Scheduling

- **Problem:** Seek time is a dominant term in DAT.

- **Disk Scheduling**
  - Given multiple outstanding disk I/O requests, what order should we dispatch them to the disk?

- **Goals:** We want a schedule that
  - Minimizes total latency (delay) for small transfers
    - e.g., a selective database query that only returns a couple records from a table
  - Maximizes throughput (bytes per unit time) for large transfers
    - e.g., returning all records in a large database table
First-Come-First-Served Policy

- Schedule each disk I/O operation in the order it arrives

**Pros**

- Fairest of them all! Requests honored as they come.
- Easy to manage pending requests with a queue (fast!)

**Cons**

- Disk-head locality is not taken into consideration
- Wide alternating arm swings are possible.
FCFS Example

- Disk head currently on track 100

- Outstanding requests for I/O to blocks on tracks:
  - 190, 55, 190, 74, 40, 190
  - (Different colors = concurrent DB transactions)

- Let’s observe a simple metric:
  - Seek time is approximated by *average seek length*
Problem with FCFS is that disk-head location is not considered

Shortest-Seek-Time-First (SSTF) Policy

1. Track current head position
2. Always issue the I/O operation that requests a track nearest in distance to the head's position
3. If there's a tie, use FCFS or random
SSTF Example

- Disk head currently on track 100
- Outstanding requests for I/O to blocks on tracks:
  - 190, 55, 190, 74, 40, 190

Evaluation

- Average seek length = ?
  - Is SSTF optimal for average seek length?
- Fairness?
  - Starvation is possible for outer tracks as requests continually arrive
Circular LOOK (C-LOOK) Policy

- How to guarantee all tracks are visited periodically?
  - Think typewriter

C-LOOK Policy

1. Just like LOOK, but head moves in only one direction
2. If no more I/Os to service in direction, move head to track 0
   (Ignoring any requests on the way back to 0)
LOOK Policy (Also called Elevator)

- Starvation of outer tracks is a problem!

**LOOK Policy**

1. Disk-head has *direction* and *current_track*
2. Sort requests in order of track number according to *direction*
3. If no more I/Os to service in *direction*, reverse direction

```
inner most track  direction  outer most track
```

`current_track`
LOOK Policy

1. Disk-head has direction and current_track
2. Sort I/O in order of track number according to direction
3. If no more I/Os to service in direction, reverse direction
LOOK Policy

1. Disk-head has *direction* and *current_track*
2. Sort I/O in order of track number according to *direction*
3. If no more I/Os to service in *direction*, reverse direction

```
current_track
```

```
Inner most track    direction    Outer most track
```
LOOK Policy

1. Disk-head has \textit{direction} and \textit{current\_track}

2. Sort I/O in order of track number according to \textit{direction}

3. If no more I/Os to service in \textit{direction}, reverse direction
LOOK Policy

1. Disk-head has \textit{direction} and \textit{current\_track}
2. Sort I/O in order of track number according to \textit{direction}
3. If no more I/Os to service in \textit{direction}, reverse direction

\begin{tikzpicture}
    \node (current_track) at (0,0) {	extit{current\_track}};
    \node (inner_most_track) at (-3,0) {	extit{Inner\ most\ track}};
    \node (outer_most_track) at (3,0) {	extit{Outer\ most\ track}};
    \node (direction) at (0,-1.5) {\textarrow{direction}};
    \draw [->] (current_track) -- (direction);
    \draw [->] (direction) -- (inner_most_track);
    \draw [->] (direction) -- (outer_most_track);
\end{tikzpicture}
LOOK Policy

1. Disk-head has direction and current_track
2. Sort I/O in order of track number according to direction
3. If no more I/Os to service in direction, reverse direction
LOOK Policy

1. Disk-head has \textit{direction} and \textit{current\_track}
2. Sort I/O in order of track number according to \textit{direction}
3. If no more I/Os to service in \textit{direction}, reverse direction
LOOK Policy

1. Disk-head has \textit{direction} and \textit{current\_track}
2. Sort I/O in order of track number according to \textit{direction}
3. If no more I/Os to service in \textit{direction}, reverse direction

\textit{current\_track}

\begin{center}
\begin{tikzpicture}
\draw[very thick,->] (0,0) -- (0,-2);
\end{tikzpicture}
\end{center}

\begin{center}
\textit{Inner most track} \quad \textit{direction} \quad \textit{Outer most track}
\end{center}
LOOK Example

- Disk head currently on track 53
- Outstanding requests for I/O to sectors on cylinders:
  - 98, 183, 37, 122, 14, 124

Evaluation
- Average seek length = ?
- Fairness and Starvation?
Topics

- Disks
  - Hardware
  - Modeling Access Time
  - Disk Scheduling

- Buffer Management
Database Architecture

**Database Admins (DBAs)**
- DDL Statements
  - DDL Compiler
- DML Statements
  - DML Compiler
  - Query Optimizer
  - Query Executor

**Casual Users**
- DDL Statements
- DML Statements

**Storage Manager**
- Buffer Manager
  - File Manager
- Transaction Manager

**Disk**
- Data Dictionary
- Indices
- Data Files

**Buffer Manager**
Fetches data from disk into memory, deciding which data to cache in memory.
Motivation: Buffer Management

- Problem:
  - Database size (on disk) is usually larger than available memory!
    - Terabytes (Disk) compared to GBs (RAM)
      ~3 orders of magnitude difference
    - To process an SQL query, we must bring file blocks into memory

- The *Database Buffer* is a segment of memory that the DB uses to *cache* some blocks for later.
  - Goal: We want to minimize number of block transfers between disk and memory when processing.
Common Block-Replacement Policies

- What does Buffer Manager do when buffer is full?

- **First In First Out (FIFO) Policy**
  - The *oldest* block that allocated in the buffer gets kicked out.
  - Pros: Fast; Simple to implement using queue
  - Cons: An *old* block doesn't mean it's hardly-ever used!

- **Least Recently Used (LRU) Policy**
  - The block that was used *furthest in the past* gets kicked out.
  - Pros: Exploits temporal locality
  - Cons: Harder to implement (uses priority queue)
Example

- Assume we only have 2 blocks in the database buffer.
- Two tables: **Book** (BNO, title, author) and **Publisher** (PNO, BNO, Name)
  - 100 Books tuples can be stored in a block
  - 50 Publisher tuples can be stored in a block
Let's run a simple select query:

```
select * from Books;
```

Access Pattern:

```
<table>
<thead>
<tr>
<th>BNO</th>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>book1</td>
<td>...</td>
<td>book100</td>
</tr>
<tr>
<td>book101</td>
<td>...</td>
<td>book200</td>
</tr>
<tr>
<td>book201</td>
<td>...</td>
<td>book300</td>
</tr>
</tbody>
</table>
```
FIFO Policy

- Database buffer allocates **two** blocks in RAM
  - 3 file blocks for the Books relation: B1, B2, B3
  - Access Pattern of `select_query()`: B1 (50x), B2 (50x), B3 (50x)

\[
\begin{array}{cccccccc}
\text{Access Pattern} & \text{B1} & \ldots & \text{B1} & \text{B2} & \ldots & \text{B2} & \text{B3} & \ldots & \text{B3} \\
\hline
\text{Buffer Slot 1} & \text{B1} & \text{B1} & \text{B1} & \text{B1} & \text{B1} & \text{B1} & \text{B3} & \text{B3} & \text{B3} \\
\text{Buffer Slot 2} & & \text{B2} & \text{B2} & \text{B2} & \text{B2} & \text{B2} & \text{B2} & \text{B2} \\
\end{array}
\]

\[\text{Number of misses to run query} = 3\]
LRU Replacement

- Database buffer allocates **two** blocks in RAM
  - 3 file blocks for the Books relation: B1, B2, B3
  - Access Pattern of `select_query()`: B1 (50x), B2 (50x), B3 (50x)

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Access Pattern</th>
<th>Buffer Slot 1</th>
<th>Buffer Slot 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B1</td>
<td>B1</td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>B1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td>B1</td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td>B1</td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>B3</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
</tr>
</tbody>
</table>
```

Number of misses = 3 (same as FIFO for this query)
Access Pattern for Nested Loop Join

<table>
<thead>
<tr>
<th>Books</th>
<th>Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNO</td>
<td>title</td>
</tr>
<tr>
<td>book1</td>
<td>pub1</td>
</tr>
<tr>
<td>book2</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>book100</td>
<td>...</td>
</tr>
<tr>
<td>book101</td>
<td>pub50</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>book200</td>
<td></td>
</tr>
<tr>
<td>book201</td>
<td>pub100</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>book300</td>
<td></td>
</tr>
</tbody>
</table>

Tuple generated: (book1, pub1) (book1, pub2) (book1, pub50) (book1, pub51) (book1, pub100)

Block: B1 P1 B1 P1 ... B1 P1 B1 P2 ... B1 P2
Access Pattern for Nested Loop Join

Books
--------------------
<table>
<thead>
<tr>
<th>BNO</th>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>book1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Publisher
--------------------
<table>
<thead>
<tr>
<th>PNO</th>
<th>BNO</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>pub1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pub50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pub51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pub100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Join with book2

Tuple:
(book1, pub1) (book1, pub2) ... (book1, pub50) (book1, pub51) ... (book1, pub100)
Block: B1  P1  B1  P1  ...  B1  P1  B1  P2  ...  B1  P2

Tuple:
(book2, pub1) (book2, pub2) ... (book2, pub50) (book2, pub51) ... (book2, pub100)
Block: B1  P1  B1  P1  ...  B1  P1  B1  P2  ...  B1  P2

Tuple:
(book100, pub1) (book100, pub2) ... (book100, pub50) (book100, pub51) ... (book100, pub100)
Block: B1  P1  ...  B1  P1  B1  P2  ...  B1  P2

100 times for B1
Access Pattern for Natural Join (Cont.)

<table>
<thead>
<tr>
<th>Books</th>
<th>Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNO</td>
<td>title</td>
</tr>
<tr>
<td>book1</td>
<td>...</td>
</tr>
<tr>
<td>book100</td>
<td>...</td>
</tr>
<tr>
<td>book101</td>
<td>...</td>
</tr>
<tr>
<td>book200</td>
<td>...</td>
</tr>
<tr>
<td>book201</td>
<td>...</td>
</tr>
<tr>
<td>book300</td>
<td>...</td>
</tr>
</tbody>
</table>

Block: B1
Tuple: (b101, p1) (b101, p2) ...

Block: B2
Tuple: (b102, p1) (b102, p2) ...

Block: B3
Tuple: (b200, p1) (b200, p2) ...

100 times for B2
FIFO Replacement Policy

- Database buffer allocates two blocks in RAM
  - Access Pattern of natural join(Book, Publisher)

<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>B1</th>
<th>P1</th>
<th>...</th>
<th>B1</th>
<th>P2</th>
<th>B1</th>
<th>...</th>
<th>B1</th>
<th>P1</th>
<th>...</th>
<th>B1</th>
<th>P2</th>
<th>B1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Slot 1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
</tr>
<tr>
<td>Buffer Slot 2</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
</tr>
</tbody>
</table>

2 misses in first 50 products:
Must fetch B1 and P2 from disk initially.

2 misses
This completes the product of tuple book1
with all tuples in Publishers

1 miss
Next 50 products for book2

2 misses
Remaining 50 products for book2
LRU Replacement Policy

- Database buffer allocates two blocks in RAM
  - Access Pattern of natural join(Book, Publisher)

<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>B1</th>
<th>P1</th>
<th>...</th>
<th>B1</th>
<th>P2</th>
<th>B1</th>
<th>...</th>
<th>B1</th>
<th>P1</th>
<th>...</th>
<th>B1</th>
<th>P2</th>
<th>B1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Slot 1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
</tr>
<tr>
<td>Buffer Slot 2</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
</tr>
</tbody>
</table>

2 misses in first 50 products:
Must fetch B1 and P2 from disk initially.

1 miss
This completes the product of tuple book1 with all tuples in Publishers.

1 miss
Next 50 products for book2

1 miss
Remaining 50 products for book2
Evaluation of FIFO vs. LRU for Natural Join

- **Full Access pattern**
  - (DAT = disk access time per block; MAT = RAM access time per block)

<table>
<thead>
<tr>
<th>Full Access Pattern for NLJ</th>
<th>No-Buffering Misses</th>
<th>FIFO Misses</th>
<th>LRU Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1, P1 (repeated 50 times)</td>
<td>50 + 50 + 50 +</td>
<td>2 + 3 + 3 +</td>
<td>2 + 2 + 2 +</td>
</tr>
<tr>
<td>B1, P2 (repeated 50 times)</td>
<td></td>
<td>3 = 150 misses</td>
<td>2 = 100 misses</td>
</tr>
<tr>
<td>...</td>
<td>50 = 5000 misses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2, P1 (repeated 50 times)</td>
<td>50 + 50 +</td>
<td>2 + 3 + 3 +</td>
<td>2 + 2 + 2 +</td>
</tr>
<tr>
<td>B2, P2 (repeated 50 times)</td>
<td></td>
<td>3 = 150 misses</td>
<td>2 = 100 misses</td>
</tr>
<tr>
<td>...</td>
<td>50 = 5000 misses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3, P1 (repeated 50 times)</td>
<td>50 + 50 +</td>
<td>2 + 3 + 3 +</td>
<td>2 + 2 + 2 +</td>
</tr>
<tr>
<td>B3, P2 (repeated 50 times)</td>
<td></td>
<td>3 = 150 misses</td>
<td>2 = 100 misses</td>
</tr>
<tr>
<td>...</td>
<td>50 = 5000 misses</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Misses for N.L.J.</strong></td>
<td>15000 misses</td>
<td>450 misses</td>
<td>300 misses</td>
</tr>
<tr>
<td><strong>Total Query Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Assume DAT = 10ms MAT = 0 ms)</td>
<td>150000 ms (= 2.5 min)</td>
<td>450 ms (= 4.5 sec)</td>
<td>300 ms (= 3 sec)</td>
</tr>
</tbody>
</table>
Reminders:

- Homework 5, Project 3 due tonight

Grading:

- Homework 4 (SQL): 25% complete

Last time...

- Magnetic disk architecture
- Files as an illusion
- Characterizing disk access time

Today...

- Tuple format (fixed vs. variable)
- Impact on query performance
Administrivia 11/4

- Grading: Homework 4 (SQL): graded
- Talk today!
- Last time...
  - Characterizing disk access time
  - Disk scheduling
- Today...
  - Buffer management
  - Tuple format (fixed vs. variable)
  - Impact on query performance