Announcements

• HW1 is due on Monday
  – How is it going?

• Readings continue to be due before class
  – How are the readings and reviews going?

• HW2 will be released by Monday

• Projects:
  – Proposals are due on Wednesday (not graded)
  – You can submit pre-proposals for early feedback (dropbox)
Where We Are

• **What we have already seen**
  – **Overview of the relational model**
    • Motivation and where model came from
    • Physical and logical independence
  – **How to design a database**
    • From ER diagrams to conceptual design
    • Schema normalization

• **Where we go from here**
  – How can we efficiently implement this model?
References


• Chapters 8 through 11 (in the R&G book, third ed.)
  – Disk and files: Sections 9.3 through 9.7
  – Index structures: Section 8.3
  – Hash-based indexes: Section 8.3.1 and Chapter 11
  – B+ trees: Section 8.3.2 and Chapter 10
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Access Methods
- Buffer Manager
- Lock Manager
- Log Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

DBMS Architecture

- Process Manager
  - Admission Control
  - Connection Mgr

- Query Processor
  - Parser
  - Query Rewrite
  - Optimizer
  - Executor

- Storage Manager
  - Access Methods
  - Buffer Manager
  - Lock Manager
  - Log Manager

- Shared Utilities
  - Memory Mgr
  - Disk Space Mgr
  - Replication Services
  - Admin Utilities

Process Model

Why not simply queue all user requests? (and serve them one at the time)

Alternatives
1. Process per connection
2. Server process (thread per connection)
   • OS threads or DBMS threads
3. Server process with I/O process

Advantages and problems of each model?
Process Per Connection

• **Overview**
  – DB server forks one process for each client connection

• **Advantages**
  – Easy to implement (OS time-sharing, OS isolation, debuggers, etc.)
  – Provides more physical memory than a single process can use

• **Drawbacks**
  – Need OS-supported “shared memory” (for lock table, for buffer pool)
    • Since all processes access the same data on disk, need concurrency control
  – Not scalable: memory overhead and expensive context switches
    • Goal is efficient support for high-concurrency transaction processing
Server Process

• **Overview**
  – DB assigns one thread per connection (from a thread pool)

• **Advantages**
  – Shared structures can simply reside on the heap
  – Threads are lighter weight than processes (memory, context switching)

• **Drawbacks**
  – Concurrent programming is hard to get right (race conditions, deadlocks)
  – Portability issues can arise when using OS threads
  – **Big problem**: entire process blocks on synchronous I/O calls
    • Solution 1: OS provides asynchronous I/O (true in modern OS)
    • Solution 2: Use separate process(es) for I/O tasks
DBMS Threads vs OS Threads

• **Why do some DBMSs implement their own threads?**
  – Legacy: originally, there were no OS threads
  – Portability: OS thread packages are not completely portable
  – Performance: fast task switching

• **Drawbacks**
  – Replicating a good deal of OS logic
  – Need to manage thread state, scheduling, and task switching

• **How to map DBMS threads onto OS threads or processes?**
  – Rule of thumb: one OS-provided dispatchable unit per physical device
  – See page 9 and 10 of Hellerstein and Stonebraker’s paper
Commercial Systems

- **Oracle**
  - Unix default: process-per-user mode
  - Unix: DBMS threads multiplexed across OS processes
  - Windows: DBMS threads multiplexed across OS threads

- **DB2**
  - Unix: process-per-user mode
  - Windows: OS thread-per-user

- **SQL Server**
  - Windows default: OS thread-per-user
  - Windows: DBMS threads multiplexed across OS threads
DBMS Architecture

- Admission Control
- Connection Mgr
- Process Manager
- Query Processor
- Access Methods
- Buffer Manager
- Storage Manager
- Log Manager
- Memory Mgr
- Disk Space Mgr
- Optimizer
- Executor
- Replication Services
- Admin Utilities
- Shared Utilities

Admission Control

- **Why does a DBMS need admission control?**
  - To avoid thrashing and provide “graceful degradation” under load

- **When does DBMS perform admission control?**
  - In the dispatcher process: want to drop clients as early as possible to avoid wasting resources on incomplete requests
    - This type of admission control can also be implemented before the request reaches the DBMS (e.g., application server or web server)
  - Before query execution: delay queries to avoid thrashing
    - Can make decisions based on estimated resource needs for a query
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Access Methods
- Buffer Manager
- Log Manager
- Lock Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

Storage Model

• **Problem**: DBMS needs spatial and temporal control over storage
  – Spatial control for performance
  – Temporal control for correctness and performance

Alternatives

• **Use “raw” disk device interface directly**
  – Interact directly with device drivers for the disks
• **Use OS files**
Spatial Control
Using “Raw” Disk Device Interface

- **Overview**
  - DBMS issues low-level storage requests directly to disk device

- **Advantages**
  - DBMS can ensure that important queries access data sequentially
  - Can provide highest performance

- **Disadvantages**
  - Requires devoting entire disks to the DBMS
  - Reduces portability as low-level disk interfaces are OS specific
  - Many devices are in fact “virtual disk devices”
Spatial Control Using OS Files

• **Overview**
  – DBMS creates one or more very large OS files

• **Advantages**
  – Allocating large file on empty disk can yield good physical locality

• **Disadvantages**
  – OS can limit file size to a single disk
  – OS can limit the number of open file descriptors
  – But these drawbacks have mostly been overcome by modern OSs
Commercial Systems

• Most commercial systems offer both alternatives
  – Raw device interface for peak performance
  – OS files more commonly used

• In both cases, we end-up with a DBMS file abstraction implemented on top of OS files or raw device interface
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Access Methods
- Lock Manager
- Buffer Manager
- Log Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

Temporal Control
Buffer Manager

• **Correctness problems**
  – DBMS needs to control when data is written to disk in order to provide **transactional semantics** (we will study transactions later)
  – OS buffering can **delay writes**, causing problems when crashes occur

• **Performance problems**
  – OS optimizes buffer management for general workloads
  – DBMS understands its workload and can do better
  – Areas of possible optimizations
    • Page replacement policies
    • Read-ahead algorithms (physical vs logical)
    • Deciding when to flush tail of write-ahead log to disk
Buffer Manager

Page requests from higher-level code

Buffer pool

Disk page

Free frame

Files and access methods

Buffer pool manager

Main memory

Disk space manager

Disk is a collection of blocks

Disk

1 page corresponds to 1 disk block
Commercial Systems

- DBMSs implement their own buffer pool managers

- Modern filesystems provide good support for DBMSs
  - Using large files provides good spatial control
  - Using interfaces like the mmap suite
    - Provides good temporal control
    - Helps avoid double-buffering at DBMS and OS levels
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Files and Access Methods
- Lock Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]
Access Methods

- A DBMS stores data on disk by breaking it into *pages*
  - A page is the size of a disk block.
  - A page is the unit of disk IO
- Buffer manager caches these pages in memory
- Access methods do the following:
  - They organize pages into collections called DB *files*
  - They organize data inside pages
  - They provide an API for operators to access data in these files
Data Storage

• Basic abstraction
  – Collection of records or file
  – Typically, 1 relation = 1 database file
  – A file consists of one or more pages

• How to organize pages into files?
• How to organize records inside a file?

• Simplest approach: heap file (unordered)
Heap File Operations

- **Create** or **destroy** a file
- **Insert** a record
- **Delete** a record with a given rid (rid)
  - rid: unique tuple identifier such that
    - can identify disk address of page containing record by using rid
- **Get** a record with a given rid
- **Scan** all records in the file
Heap File Implementation 1

Linked list of pages:

Header page

Data page

Data page

Data page

Full pages

Pages with some free space
Heap File Implementation 2

Better: directory of pages

Header page

Directory

Directory contains free-space count for each page.
Faster inserts for variable-length records
Issues to consider

- 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
  - Fixed length
  - Variable length
- Record id = RID
  - Typically RID = (PageID, SlotNumber)

Why do we need RID’s in a relational DBMS?
See discussion about indexes later in the lecture
Page Format Approach 1

Fixed-length records: packed representation

Problems?
How to handle variable-length records?
Need to move records for each deletion, changing RIDs
Can handle variable-length records
Can move tuples inside a page without changing RIDs
Record Formats

Fixed-length records → Each field has a fixed length (i.e., it has the same length in all the records)

| Field 1 | Field 2 | ... | ... | Field K |

Information about field lengths and types is in the catalog
Record Formats

Variable length records

Remark: NULLS require no space at all (why ?)
Long Records Across Pages

- When records are very large
- Or even medium size: saves space in blocks
- Commercial RDBMSs avoid this
LOB

• Large objects
  – Binary large object: BLOB
  – Character large object: CLOB

• Supported by modern database systems
• E.g. images, sounds, texts, etc.

• Storage: attempt to cluster blocks together
Types of Files

- **Heap file** *(what we discussed so far)*
  - Unordered
- **Sorted file** *(also called sequential file)*
- **Clustered file** *(aka indexed file)*
Modifications: Insertion

• File is unsorted (= heap file)
  – add it wherever there is space (easy 😊)

• File is sorted
  – Is there space on the right page?
    • Yes: we are lucky, store it there
  – Is there space in a neighboring page?
    • Look 1-2 pages to the left/right, shift records
  – If anything else fails, create overflow page
Overflow Pages

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

• Free space in page, shift records
  – Be careful with slots
  – RIDs for remaining tuples must NOT change

• May be able to eliminate an overflow page
Modifications: Updates

• If new record is shorter than previous, easy 😊
• If it is longer, need to shift records
  – May have to create overflow pages
Searching in a Heap File

File is **not sorted** on any attribute

**Student(sid: int, age: int, ...)**

```
30 | 18 ...
70 | 21
```

1 record

```
20 | 20
40 | 19
```

1 page

```
80 | 19
60 | 18
```

```
10 | 21
50 | 22
```
Heap File Search Example

• 10,000 students
• 10 student records per page
• Total number of pages: 1,000 pages
• Find student whose sid is 80
  – Must read on average 500 pages
• Find all students older than 20
  – Must read all 1,000 pages
• Can we do better?
Sequential File

File sorted on an attribute, usually on primary key

Student(sid: int, age: int, ...)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>18</td>
</tr>
<tr>
<td>40</td>
<td>19</td>
</tr>
<tr>
<td>50</td>
<td>22</td>
</tr>
<tr>
<td>60</td>
<td>18</td>
</tr>
<tr>
<td>70</td>
<td>21</td>
</tr>
<tr>
<td>80</td>
<td>19</td>
</tr>
</tbody>
</table>
Sequential File Example

• Total number of pages: 1,000 pages
• Find student whose sid is 80
  – Could do binary search, read $\log_2(1,000) \approx 10$ pages
• Find all students older than 20
  – Must still read all 1,000 pages
• Can we do even better?