Wrap-up and Conclusions

Looking Back

• Recall our two-layer view of a DBMS:

• When choosing an approach to information management, choose an option for each layer.

• We’ve seen several options for the storage layer:
  • transactional storage engine
  • flat files (XML text docs, JSON docs)
  • native XML DBMS
  • NoSQL DBMS (with support for sharding and replication)

• We’ve also looked at several options for the logical layer:
  • relational model / object-relational model
  • semistructured: XML, JSON
  • other NoSQL models: key/value pairs, column-families
One Size Does *Not* Fit All

- An RDBMS is an extremely powerful tool for managing data.
- However, it may not always be the best choice.
  - can be overkill if don’t need all of its features
  - can be hard / expensive to setup / maintain / tune
  - may not provide the necessary functionality
    - examples: data mining, queries on genomic data
  - footprint may be too large
    - example: for a small embedded system
  - may be unnecessarily slow for some tasks
    - overhead of IPC, query processing, etc.
  - does not scale well to large clusters

One Size Does *Not* Fit All (cont.)

- Other applications for which an RDBMS is not the best solution:
  - data warehouses
  - processing streams of data (e.g., real-time financial data)
  - search engines

- See the paper from Stonebraker and Centitemel on the course website.

- Need to learn to choose the right tool for a given job.

- In some cases, may need to develop new tools!
Implementing a Storage Engine

• We looked at ways that data is stored on disk.

• We considered index structures.
  • B-trees and hash tables
  • provide efficient search and insertion according to one or more key fields

• We also spoke briefly about the use of caching to reduce disk I/Os.

Implementing a Transactional Storage Engine

• We looked at how the “ACID” properties are guaranteed:
  
  Atomicity: either all of a txn’s changes take effect or none do
  Consistency preservation: a txn’s operations take the database from one consistent state to another
  Isolation: a txn is not affected by other concurrent txns
  Durability: once a txn completes, its changes survive failures
Distributed Databases and NoSQL Stores

• We looked at how databases can be:
  • fragmented/sharded
  • replicated

• We also looked at NoSQL data stores:
  • designed for use on clusters of machines
  • can handle massive amounts of data / queries
  • relax transactional guarantees for increased performance and availability

Logical-to-Physical Mapping

• The topics related to storage engines are potentially relevant to any database system.
  • not just RDBMSs
  • any logical layer can be built on top of any storage layer

• Regardless of the model, you need a logical-to-physical mapping.
  • from the logical schema of a collection of data
  • to its physical representation.

• In the problem sets, you’ve implemented a logical-to-physical mapping for the relational model using Berkeley DB.

• We also looked at options for how to perform this mapping for XML.
Conclusions

• One size does not fit all!

• Database system = logical layer + storage layer

• We’ve studied options for both layers, and how to map between them.

• Understanding these topics should help you to:
  • choose logical and physical layers for a given task
  • tune existing systems for better performance
  • potentially develop new tools
    • with new combinations of layers
    • and/or with more efficient mappings

Final Exam Details

• Monday, December 16 from 9:00-11:00 in KCB 101
• two hours long
• one 8.5 x 11 sheet of notes
• entire semester is fair game
• two parts:
  1. multiple choice (3 pts each, 1 pt. for second choice)
  2. short answer: similar to the practice exercises
     • there will be some choice here
**Tree-Based Index Structure Problem**

- Consider the following tree-based index structure, in which the keys are a person's last name:

```
  Baker  Ford  Li  Patel
   /     /    /    /
Abel  Abel Arthur Baker Cho Guo Hicks Ho Li Ma Ng North Smith Zhou
```

- Is this a B-tree or a B+tree?

- If the tree has an order of 2, what would it look like after inserting a key/value pair for a person whose last name is Partridge?

**Two-Phase Locking Problem**

- Consider the following schedule:

```
<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>w(A)</td>
<td>w(A)</td>
</tr>
<tr>
<td></td>
<td>r(C)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>w(C)</td>
</tr>
</tbody>
</table>
```

- Is it possible under two-phase locking? If so, insert lock and unlock operations as needed.

- Is it possible under strict two-phase locking?
Logging and Recovery Problem

• Recall that we studied three logging schemes:
  • undo-redo
  • undo-only
  • redo-only

• What type of logging is being used to create the log at right?

<table>
<thead>
<tr>
<th>LSN</th>
<th>record contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>txn: 1; BEGIN</td>
</tr>
<tr>
<td>150</td>
<td>txn: 1; item: D1; new: 2500</td>
</tr>
<tr>
<td>350</td>
<td>txn: 2; BEGIN</td>
</tr>
<tr>
<td>400</td>
<td>txn: 2; item: D2; new: 6780</td>
</tr>
<tr>
<td>470</td>
<td>txn: 1; item: D1; new: 2750</td>
</tr>
<tr>
<td>550</td>
<td>txn: 1; COMMIT</td>
</tr>
<tr>
<td>585</td>
<td>txn: 2; item: D1; new: 1300</td>
</tr>
</tbody>
</table>

original values:
D1=1000, D2=3000

• What are the possible on-disk values of item D1 given that logging scheme and this log?

• What are the possible on-disk values of D2?

ACID Properties Problem

• Recall that a transaction has the following “ACID” properties:
  Atomicity: either all of its changes take effect or none do
  Consistency preservation: its operations take the database from one consistent state to another
  Isolation: it is not affected by other concurrent transactions
  Durability: once it completes, its changes survive failures

• If you disable locking in your DBMS, which of these properties do you lose?

• What if you disable logging?

• What other properties do we need to ensure full isolation?
SQL Problem

- Recall our movie database from PS 2:
  Movie(id, name, year, rating, runtime, genre, earnings_rank)
  Person(id, name, dob, pob)
  Actor(actor_id, movie_id)
  Director(director_id, movie_id)
  Oscar(movie_id, person_id, type, year)

- Write a query to retrieve the winners (name and type) of the "big 5" Oscars (everything but Best Director) in the year 2000.

Deadlock Detection Problem

- Would the following schedule produce a deadlock under rigorous 2PL? Assume that no commits occur during the sequence of operations shown.
  r_4(B); r_1(B); r_3(C); r_2(C); r_6(D); w_2(B); r_1(A); w_5(A); w_6(C); w_3(D); …

```
T1   T2   T3
  T4   T5   T6
```
ER Diagram Problem

• Consider the following ER diagram:

```
ER Diagram
A -> R <- B
aid -> name
bid -> name
```

• What constraints does this diagram specify?

• What attributes would be needed for the primary key of the relation used to capture R?

XML Problem

• Consider an XML document that looks like this:

```
<students>
  <student>
    <name>Alan Turing</name>
    <course>CSCI E-119</course>
    <course>CSCI E-268</course>
    <course>CSCI E-50a</course>
    <course>CSCI E-50b</course>
    <course>CSCI E-160</course>
    <course>CSCI E-215</course>
    <course>CSCI E-113</course>
  </student>
  ...
</students>
```

• Give two different ways of finding the names of all students taking CSCI E-268.
XML Problem (cont.)

• Here’s one XPath expression that would work:

• Here’s one FLWOR expression that would work:

---

MongoDB Problem

{ _id: "0499549",
 name: "Avatar",
 year: 2009,
 rating: "PG-13",
 runtime: 162,
 genre: "AVYS",
 earnings_rank: 1,
 actors: [{ id: "0000244", name: "Sigourney Weaver" }, ... ],
 directors: [{ id: "0000116", name: "James Cameron" }] }  

• Write a query to find, for each director of a top-200 movie, the number of top-grossing movies by that director.
Timestamp-Based Concurrency Control Problem

- How will this schedule be executed (without commit bits)?
  \[w_1(B); r_2(A); w_2(A); r_1(A); r_3(C); w_3(B); w_2(B); w_2(C); r_3(A); w_3(C)\]

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>RTS = WTS = 0</td>
<td>RTS = WTS = 0</td>
<td>RTS = WTS = 0</td>
</tr>
</tbody>
</table>

Replication Problem

- A database is being replicated across 20 sites, and a client wants to update one of the data items.

- If the database is being managed by MongoDB, how many of the copies would a client need to update?

- If the database is being managed by a system that uses synchronous replication and fully distributed locking, what is the minimum number of copies that a client would need to update?
Object-Relational Problem

• How could we modify the schema of the movie database to take advantage of the features of the object-relational model? For the sake of this problem, you should assume that each movie has a single director.

• Write a query to find the 12 top-grossing movies and the names of their directors. The result should be tuples with the earnings rank, movie name, and director's name.