
Overview of lectures

February 6, 2003

Advanced Database Technology, Spring 2003

— Lecture 2: Data storage devices. I/O model.

When dealing with large data sets one needs storage outside of the computer's main memory, e.g.:

- Disks.
- Tapes.
- The memory of other computers.

The access time for a given byte on such devices is *thousands or millions* of times larger than retrieving a byte in main memory.

Lecture 2 concerns the characteristics of external memory devices. Such an understanding is necessary to devise efficient external memory algorithms.

— Lecture 2, cont. —

Block transfers:

Because of the large access time, every external memory access is used to transfer a whole *block* of adjacent data.

- E.g., disk blocks are typically 4-16 kilobytes.

Question: Why do block transfers help?

A particularly simple model of external memory is the *I/O model*, which will be used for most of the material in the course.

As a primer, we consider efficient *sorting* in the I/O model.

— Lecture 3: Representing data elements. —

In lecture 3 we ask: How does one store relations in a blocked memory?

<i>title</i>	<i>year</i>	<i>length</i>	<i>filmType</i>
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

Schema:Movie; Star Wars;
1977; 124; color; Mighty
Ducks; 1991; 104; color

Schema:Movie; Wayne's
World; 1992; 95; color;
...

Problems with updates:

What if we want to add “Episode 4” to “Star Wars” but there is not sufficient space in the block?

— Lecture 4: Index structures. —

Relational operations can sometimes be computed much faster if we have precomputed a suitable data structure.

Most notably, two kinds of so-called *index structures* are essential to database performance:

- B-trees.
- External hash tables.

For example, hash tables may speed up relational operations that involve finding all occurrences in a relation of a particular value.

Lecture 4 concerns *how* such index structures work, *why* they are useful, and what the *price* paid is.

— Lectures 5 & 6: Impl. of relational operations.

We consider the question: Given a query in e.g. SQL, how can the result be efficiently computed in the database?

One problem in answering this is that the best way of doing things may depend crucially on the data in the relations.

Example:

<i>ID</i>	<i>Name</i>
1	P. Persson
2	J. Jensen
3	P. Schlüter
...	...
1447	S. Isted

 \bowtie

<i>ID</i>	<i>HSAS</i>	<i>Mark</i>
1	ADBT	n/a
1	DSK	7
2	WEB	10
...
1447	OOP	n/a

 \bowtie

<i>ID</i>	<i>Project</i>	<i>Done</i>
211	XML	n/a
347	XML	n/a
790	XML	n/a

— Lecture 7: Recovery from system failures. —

Things go bad...



... and we want to be able to recover.

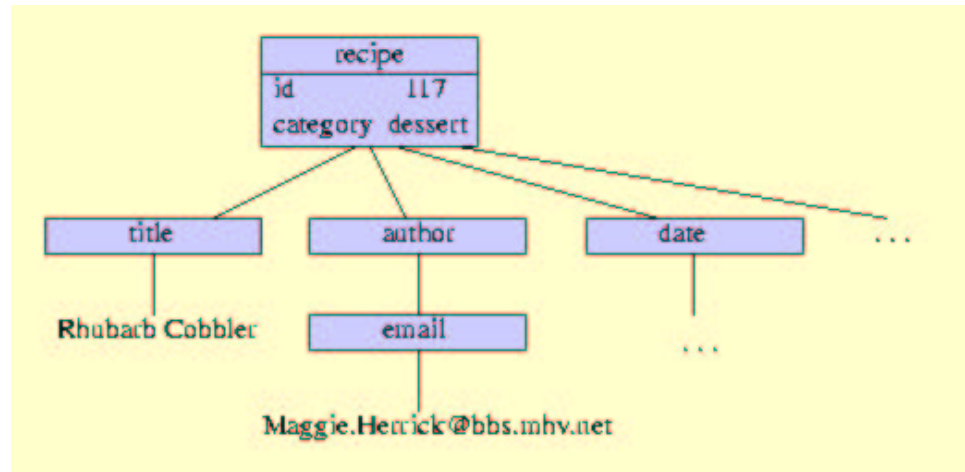
How do we cope with system failures, e.g.:

- disk crash in the middle of a transaction?
- power failure during recovery?

— Lecture 8: XML in databases. —

Guest lecturer: Albrecht Schmidt, Aalborg University.

XML is a `<hot>` textual format for tree structured data.



Large XML data sets should be stored in databases for efficiency.

- With tree data, new basic operations become relevant (XM-algebra?)
- Databases that directly support such operations are in their infancy.

We expect this lecture to concern mainly how to embed XML data, and corresponding query languages, into relational databases.

— Lecture 10: Geometric databases. GIS. —

Guest lecturer: Joachim Gudmundsson, Eindhoven University.

Many large data sets are geometric in nature, e.g., satellite images and map data (road networks, altitude maps, etc.)

Examples of geometric/GIS queries:

- Show a specific part of a map.
- Show the 10 gas stations nearest to your location.
- Show the shortest path to some destination.

This lecture will introduce some of the problems in geometric databases, and some of the most important tools for dealing with such problems.

— Lecture X: Text indexing. —

Searching in large amounts of text has gained importance due to the success of the World Wide Web.

For example, the search engine **Google** facilitates rapid keyword searches in over 3 billion web pages.

- Possible only due to a huge precomputed data structure that is regenerated a few times a year.

This lecture introduces basic tools in (large scale) text indexing, in particular *suffix trees* and *suffix arrays* (on secondary memory).

— Lecture Y: Cache-oblivious algorithms. —

The block transfers between main memory and disk are not the only ones. Block transfers also occur, e.g.:

- Between the processor cache and main memory.
- Between different levels of processor cache.
- In network communication.

This lecture introduces *cache-obliviousness*, a new approach to designing algorithms that are simultaneously efficient on all levels of the above-mentioned *memory hierarchy*.

— Lecture Z: Data streaming. —

OR: “Anti-databases”

In some applications, e.g. network monitoring, new data comes in a steady stream. Using traditional databases to query such data sets poses several problems:

- The data sets can be too large to store.
- The time required to make queries is too large for real-time applications.

Data stream algorithms answer pre-defined “queries” about a stream of data, using much less memory than that required to store the entire stream.

This lecture presents some basic algorithms on data streams, and considers limits on what can be done by such algorithms.