
Advanced Database Technology
Rasmus Pagh and S. Srinivasa Rao
IT University of Copenhagen
Spring 2006

Representing data elements

February 13, 2006

Based on Chapter 12 in GUW, [Pagh03] Sec. 1, and [CLRS01] pp. 405-409

— This lecture: Representing data elements. —

In this lecture 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?

— Overview of this lecture —

- Storing fixed sized tuples
- Dealing with pointers
- Variable length tuples
- Updates
- Queues, stacks, and linked lists (I/O model and amortized analysis)
- Index structures (separate slide set)

— Some terminology —

- **Attributes** are stored as a sequence of bytes, called **fields**.
- **Tuples** are stored as a collection of fields, called **records**.
- Records are put together and are stored in **blocks**.
- A **relation** is a collection of records stored in blocks, called a **file**.

— Attributes stored in fields —

The schema of a relation specifies the type of attributes. This decides how much space is needed to store a relation. (It may be of variable size.)

- CHAR(7) a string of length 7 is stored in 7 bytes.
- BIT(2) is two bits, can be stored in two bits, but often a whole byte is used.
- {RED, GREEN, BLUE, YELLOW} is an enumerated type that can be stored as 00, 01, 10, 11, i.e. two bits is enough.

— Variable sized data —

Some attributes may not have fixed sized. If the size varies a lot for different tuples, then we do not want to allocate memory for all tuples to be able to store the maximum sized attributes.

However, that is what `VARCHAR(n)` in SQL does. $n + 1$ is the number of bytes allocated for the string, even if it may be shorter.

Two solutions:

- **Length + content:** $n + 1$ bytes allocated for a string of length n .
6 S t r i n g (assuming $n < 256$)
- **Null-terminated string:** $n + 1$ bytes allocated for a string of length n .
S t r i n g ⊥

Records

A tuple is stored in a record. The size is the sum of sizes of the fields in the record. A record often also stores a 'header'.

A record header might store information such as

- the schema for the record (or a pointer to it)
- Size of the record
- Timestamps (last read, last updated)

The schema is used to access specific fields in the record.

— Schema information —

Tells us how the fields (attributes) are stored within a record (tuple).

It contains

- the attributes of the relation
- their types
- the order in which attributes appear in a tuple
- constraints on the attributes and the relation

— Fixed-length records in blocks —

Records are stored in blocks. Typically a block only contains one kind of records.

The block may have a header with info.:

- Index information, often in form of pointers. (More on indexes later today.)
- Type of tuples in the block.
- Offset table for the records in the block. Needed if records are of variable length.
- Block ID.
- Timestamps.

— Problem session: Packing fields —

Read the box on page 573 in the book and discuss:

- Do you agree with their conclusion?
- When is it a good/bad idea to pack fields?

— Block and record addresses —

Addresses (pointers) to fields, records and blocks are often part of records and we have to deal with them in a special way. E.g., pointers to schemas and pointers used in index structures are stored in records.

Why addresses are different from other kind of data:

- Blocks are moved from secondary memory to main memory when they are used.
- Records may move, both within a block and from one block to another.
- Records may be deleted.
- Attribute values may change size, i.e. data move within a record.

— Block addresses in main and secondary memory

Block address for blocks in main memory:

The block has an internal memory address when it is loaded into a buffer in main memory.

Block address for blocks in secondary memory:

The physical address has to be used. The physical address describes the physical location of the block.

— Physical and logical addresses —

Physical address

Describes physical location, i.e. which disk, which cylinder, which track etc.
Typical size is 8-16 bytes.

Logical address

A fixed length arbitrary string for each record. A *map table* is used to map logical addresses to physical addresses.

Useful when records are moved, since only the map table has to be updated, and not the references to the record.

— Structured addresses —

Structured address

A combination of physical and logical addresses. E.g., only the physical address for the block. To find a record, an offset table in the block or another kind of search in the block is needed.

Reasons why structured addresses are useful:

- A record can move within a block and still have the same structured address.
- When a record is removed it can be replaced by a **tombstone** that marks it deleted. The structured address can still be unchanged. When the record is looked up we know it is deleted.

Offset tables

How to organize an offset table:

- Grow the offset table from left to right and insert records from right to left (since we do not know the size of the offset table when dealing with variable length records and when using tombstones).
- If the entries of the offset table are large enough references to other blocks can be stored. Useful when records are moved and we do not want to update the address.
- The tombstone can be stored in the offset table and the space used by the deleted record can be reused by another record.

— Pointer swizzling —

How to manage pointers when blocks are moved between main memory (memory addresses) and secondary memory (database addresses).

- When in secondary memory, database addresses are used.
- When in main memory, database or memory addresses may be used.

Using memory addresses is more efficient. Otherwise translation is needed.

A translation table is used to map database addresses to memory addresses.

Pointer swizzling

When a block is moved from secondary to main memory, pointers in the block can be swizzled (translated) from database addresses to memory addresses. A bit indicates the type of address.

— Swizzling strategies —

Automatic Swizzling

When a block is moved into main memory, all pointers in the block are swizzled if possible. All addresses to blocks currently in memory are stored in the translation table.

Swizzling on Demand

Pointers are swizzled when they are followed. When a block is moved into memory only the translation table is updated.

No Swizzling

Pointers are never swizzled. The translation table is used all the time.

— Problem session: Swizzling —

- Discuss the pros and cons of the three swizzling strategies:
 - Automatic Swizzling,
 - Swizzling on Demand, and
 - No Swizzling.

When is it a good/bad idea to use them?

- What are the problems when a block is written back to disk? And how can they be solved?

— Variable-length data and records —

Reasons why records not always have the same size:

- Fields of variable length. Attribute content vary in size.
- Repeating fields. An attribute that appears several times, but how many times is not specified by the schema.
- Records of variable format. When different tuples in a relation have different sets of attributes. E.g., if many attributes have no content.
- Enormous fields. Data like movies and pictures in the relation. The record may not fit into one block.

— Fields of variable length —

When a field has variable size we still have to be able to find all fields in the record. Since the offset cannot be read from the relation schema some extra information is stored in the record header.

Example of how it can be solved:

- Store fixed length fields first in the record.
- Store the total size of the record.
- Store offsets for variable sized fields (except the first).

— Repeating fields —

When a record contains a variable number of a field.

Store information in the record header to locate all occurrences of the field in the record.

A method to deal with fields of variable size and variable number of occurrences:

- Keep the record fixed size.
- Store variable length data in a separate block and use a pointer to it.
- Fixed sized records can be searched more efficiently. Less information is needed in the header. Moving records is easier.
- The number of I/O's increase, since a pointer has to be followed.

Mixed strategies may be a good solution.

Spanned records

A record is called **spanned record** if it is split between two or more blocks.

Reasons for spanned records:

- Space utilization.
- Records larger than a block.

For each fragment of a record, extra information on where to find next and previous fragment is needed.

BLOBS

Binary, Large ObjectS = BLOBS

BLOBS can be images, movies, audio files and other very large values that can be stored in fields.

Storing BLOBS

- Stored in several blocks.
- Preferable to store them consecutively on a cylinder for efficient retrieval.

Retrieving BLOBS

- A client retrieving a movie may not want it all at the same time.
- Retrieving a specific part of the large data requires an index structure to make it efficient.

— Problem session: Updates —

We will look at three types of updates:

- Insertions of new tuples
- Deletions of tuples
- Tuple updates

What problems may arise when updates are performed on the database?

Think of the different situations where we have:

- fixed length vs. variable length tuples
- no order vs. sorted tuples

Updates

Insert

No order: No problem, just find a block with enough space or use a new block.

Fixed order: May be a problem if there is not enough room in the correct block. Solutions:

1. Find space in nearby block and rearrange
2. Create an overflow block

Delete

Pack data in the block to prepare for new inserts. Remove overflow blocks if possible. Leave a tombstone if there may be pointers to the record.

Update

Fixed length: No problem.

Variable length: Same as for insert and delete. (But no tombstones.)

— Stacks and Queues —

A **stack** maintains a collection of items in which only the most recently added item may be removed.

A **queue** maintains a collection of items in which only the earliest added item may be accessed/removed.

How can we maintain a stack or queue in external memory?

– use *buffering*

“macroscopic view” in external memory is same as “microscopic view” in internal memory.

— Problem session on linked lists —

Summary

- Storing fixed sized tuples
- Variable length tuples
 - offset tables
 - overflow blocks
- Dealing with pointers
 - logical and physical addresses
 - database and memory addresses
 - pointer swizzling
- Updates
- stacks, queues and linked lists in external memory