Storing Relational Data on Disk

Disk Blocks

Recall:

- disk access operations are more expensive than data processing in main memory;
- an individual Read or Write command issued to a disk controller can retrieve information stored in a number of consecutive *disk sectors*.

To ensure that data stored in databases is retrieved efficiently, DBMS choose the following data storage approach:

- A **block** or **disk page** of a specific size is selected. The size of a block cannot be too large — this may lead to wasted space, but should not be too small — this will increase the number of disk access operations.
- **Each relational table, and all supplemental index structures are stored as collections of blocks/pages on disk**.

Standard block sizes are 2, 4, 8, 16Kb. Larger blocks are used less often.

Storing Relational Data in Disk blocks.

Note, the the solution proposed below is by far, not the only possible. There may be a lot of different ways to store relational data on disk, e.g., grouping attributes together, using **only** index structures, storing data from different tables in the same file, etc... As we discuss the traditional storage techniques, it will become apparent why they are used and are considered efficient.

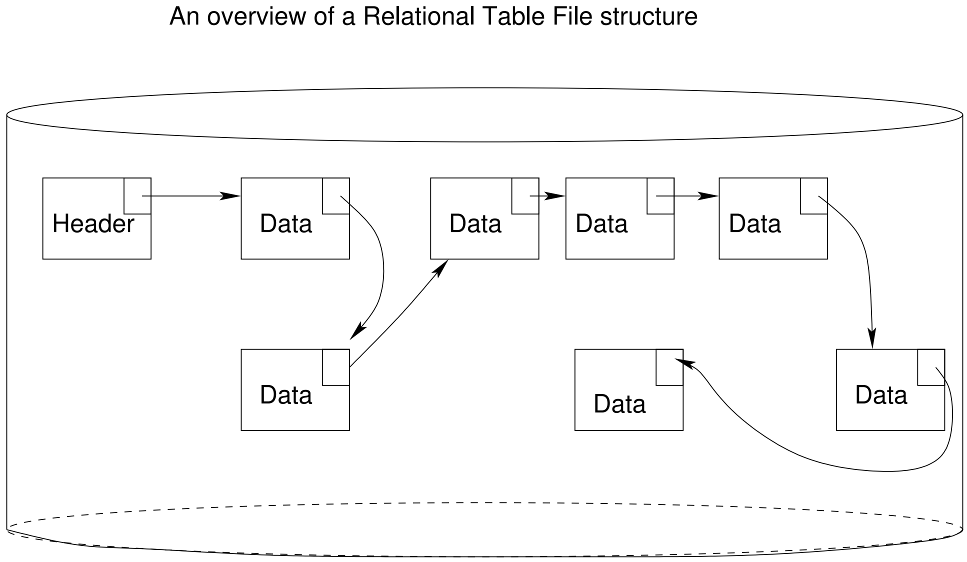

Figure 1: Database files on disk.

Each relational table is stored on disk as a single file, broken into a sequence of disk pages. Individual blocks may be located in different places on disk (different surfaces, tracks, cylinders, etc...), but all content within a single page is a consequtive seqence of bytes.

A *disk block* is typically equal to one or more *sectors* of the disk. Note that *sector* is a term that describes physical properties of the disk, while *block* and *page* describe logical constructs.

A typical **relational table file** consists of

1. Header page, the first page of the file.
2. Data pages, all remaining pages in the file.

Occasionally, depending on the types of data stored in the relational table, other types of disk pages may be present in the file as well. The structure of a relational table file is shown on Figure 1.

Header page

A header page is the first page/block of any database file. This page does not contain any relational data from the database. Rather it contains useful information about the file itself, as well as meta-information about the relational table.

The following information can be stored on the header page:

- Format identification information (something that tells DBMS, "I am your file");
- Relational table schema information;
- Record structure information: sometimes record structure on disk is different than the logical structure of the table.

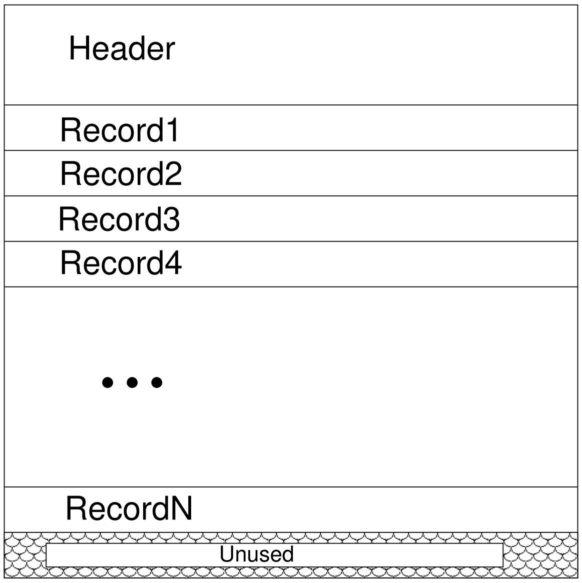

Figure 2: Data Page/Block structure in a nutshell

- Record size;
- Starting points/pointers for various linked lists within the table file:
  - Full ordered list of blocks;
  - List of blocks with open space;
- Indexing information for *non-heap* database files (we will look at how records are organized in the file below);
- Various timestamps.
- etc...

Generally speaking, the size of the block is typically more then enough to accommodate any information DBMS designer finds useful to store in header pages.

Data page

Figure 2 shows the structure of a data block. It consists of

1. *Block header*, a sequence of bytes (typically at the beginning of the page) that stores information about the current state of the disk page and any linked list pointers for the entire file.
2. *Data records*: the main portion of the disk page is broken into *records*, sequences of bytes storing data from a single row or the relational table.
3. *unused space*: any leftover space that cannot be used to store a full record.

Storing Data in Disk Records

Each record stores information about a single tuple. It is broken into parts representing values of each individual attribute of the tuple. In addition, records may contain some extra information.

Representing Attributes

Attribute Type	Storage Requirements
`INTEGER`	2 or 4 bytes
`FLOAT`	4 or 8 bytes
`CHAR(n)`	array of n bytes; unused bytes marked with $\perp$ ("pad") character
`VARCHAR(n)`	*Length+content*: array of $n+1$ bytes; first byte holds # bytes in string, remaining bytes hold string content; *Null-terminated string*: array of $n+1$ bytes, filled with string characters, terminated by *null* character
`BIT(n)`	array of $(n\mathtt{div}8)+1$ bytes
Enumerated types	Map values to integers, store as `INTEGER` value
`DATE, TIME`	Converted into `INTEGER`

Grouping Attributes in Records

Most DBMS use records of *fixed size*. Here we will concentrate on such records. Other applications, e.g., Information Retrieval, require records of varying size. These will be discussed separately later.

Building Fixed-size Records

First approach to building a record is to *concatenate the represenations of the tuple's attributes (a.k.a., fields) together*. However, in doing so, the following rules **must be observed**:

- *INTEGER* values (for 4-byte integers) must start at positions divisible by 4.
- *FLOAT* values (for 8-byte floating point numbers) must start at positions divisible by 8.
- Sometimes, the rule is that all other fields must start at positions divisible by 4.

If there are fields of sizes not divisible by 4, or if concatenation leads to INTEGERs and FLOATs starting at wrong positions, the following can be done to rectify the situation:

- Padding: empty, unused bytes are added between the fields to ensure that the next field starts at the right position/offset.
- Field reodering: the record is built by fist putting all FLOAT values, then all INTEGER values, then all other values. Because this structure may differ

from the order of attributes specified by `CREATE TABLE` statement, the new order of attributes needs to be recorded somewhere (e.g., in the header page of the file).

Record Headers

In addition, records can contain *headers*. The choice of whether to include a record header, and what information to put in it is up to the DBMS designer. Record headers for fixed-size records can contian the following information:

- Record schema/pointer to the place in the file where record schema is stored (may also be contained in the block header).
- Length of the record.
- Timestamps for record modification/access
- Record state (active/deleted), a.k.a, "tombstone".

Block Headers

Block Headers typically store information about the current state of the block, and the block's "position" in the overall file. The information stored in the header is determined by the DBMS designer. Typically, it may include the following:

- Block ID;
- Links to other blocks in the file:
  - next/previous block in the block order in the file;
  - next/previous block that has space avaialble for new records;
  - next/previous block in an special indexing order;
- Information about the relation, relational schema/pointer to the schema.
- General information about records in the block: record size, total number of slots, number of used records, etc...
- Information about availble record space on the block:
  - Number of available slots/records;
  - Record availability bitmap;
- Timestamps.