Physical Access to Data

Physical access structures describe how data is stored on disk; in order to provide the best query execution is very important know which can be the right structure.

The access method manager performs:

. the order in which executions chosen by the optimizer have to be done;

. the sequence in which physical access to disk pages are done.

This job is realized with access methods: they are software modules specialized for a single data structure and they provides primitives for ready/writing data.

The organization of a disk page is different for different access methods; it can divided into:

. space for data;

. space for access method control information;

. space for file system control information.

Physical access structures

Data storage can be done:

. in sequential structures;

. in hash structures.

In order to increase performances is possible have index:

. trees structures;

. unclustered hash index.

Sequential structures

The way in which data is stored is sequential; there are two possibles realizations:

. heap files;

. ordered sequential structure.

Heap file

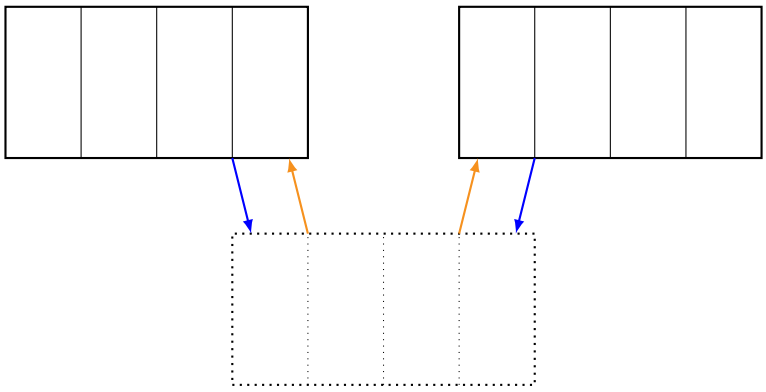
The insertion is made *appending* tuples at the end of the file. The compact representation is useful if there are few reads but is very difficult to sort. Having a compact representation means that a new block of memory is started only if the previous is full up.

In order to read a tuple DBMS have to read all blocks until the tuple is reach.

Ordered sequential structure

Tuples are written on blocks in order specify by the value of an attribute, called *sort key*. The sort key is very important because if the read of the table is done through another attribute the structure provides the same performances than the heap file.

In this method is very important to maintain the order when new tuples are inserted: there are three possibles solutions. The first one is *overprovisioning* the space: it means that when table is created some free space is leave between two blocks. The second method consist of *copy* a block in main memory and doing a resorting while the third solution is *overflow files* (shown below in the picture).



This structure is tipically used with B^+-trees indices where the index key is the sort key.

If the operation have to be done is very strong, is possible that work can be divided in *sub-operations* and, when they are processed, DBMS can use ordered sequential structure to store intermediate results until they are merged.

Indices based on trees are made of two parts: the index key and the value who points to the entire tuple.

The structure of a tree index includes:

. a **root** node;

. some **intermediates** nodes;

. **leaf** nodes.

Data is accessed by leaf node in two different ways:

. if the leaf node correspond to the physical memory block, the index is called *clustered*;

. if memory blocks and index are indipendent the approach is called *unclustered*.

For their properties, on a table is possible to have only one clustered index and more than one unclustered index.

B^+-trees are balanced trees so all leaves have the same distance from the root. The different between B-trees and B^+-trees is that in the first one leaves are not linked together. Nowadays B^+-trees indices are more used.

The principal disadvantage having index on a table is that operations like update the table, insertion or remove tuples must also be done synchronously on the index.