Databases 3

Elements of Data Science and Artificial Intelligence

Prof. Dr. Jens Dittrich

bigdata.uni-saarland.de

January 23, 2020

Induced Horizontal Partitioning

Induced Horizontal Partitioning

Given a partitioning function $p : [R] \rightarrow D$, the horizontal partitioning of R into partitions R_i such that $\forall_{t \in R} t \in R_{p(t)}$ is called the *induced horizontal partitioning*.

Examples:

$p_0 : [R] \rightarrow int, p_0(t) := t.a$ modulo 2

- $p_0((2, A)) = 0$
- $p_0((7, B)) = 1$
- $p_0((1, B)) = 1$
- $p_0((6, C)) = 0$

Induced Horizontal Partitioning:
$R_0 = \{(2, A), (6, C)\}$,
$R_1 = \{(7, B), (1, B)\}$

$p_1 : [R] \rightarrow char, p_1(t) := t.b$

- $p_1((2, A)) = A$
- $p_1((7, B)) = B$
- $p_1((1, B)) = B$
- $p_1((6, C)) = C$

Induced Horizontal Partitioning:
$R_A = \{(2, A)\}$, $R_B = \{(7, B), (1, B)\}$,
$R_C = \{(6, C)\}$

Induced Recursive Horizontal Partitioning

Induced Recursive Horizontal Partitioning

Let $p_0, p_1, \ldots, p_{ml}$ be partitioning functions with $p_l : [R] \rightarrow D_l \ \forall_{0 \leq l \leq ml}$.
Initial Step: we apply p_0 on R. This yields a set of horizontal partitions $\{R_{l_0}\}, l_0 \in D_0$.
Recursion Step: we iteratively apply p_{l+1} on the horizontal partitions created by the previous partitioning step $p_{l \geq 0}$.
Set of induced partitions: This yields a set of horizontal partitions $\{R_{l_0,\ldots,l_{ml}}\}, l_i \in D_{l_i}$.

Example:

$p_0 : [R] \rightarrow int, p_0(t) := t.a$ modulo 2 $\qquad$ $p_1 : [R] \rightarrow char, p_1(t) := t.b$

Induced Recursive Horizontal Partitioning:
$R_{0,A} = \{(2, A)\}$, $R_{0,C} = \{(6, C)\}$,
$R_{1,B} = \{(7, B), (1, B)\}$

Index

Index

Any set of (recursive or non-recursive) horizontal partitioning functions that can be exploited by a query to reduce its search space is called an *index* (or index structure).

More concretely, the goal of an index is to reduce the set of horizontal partitions we have to inspect to compute the result for a query.

Examples:

$[R] = \{[a : int, b : char]\}$

$R_{0,A} = \{(2, A)\}$, $R_{0,C} = \{(6, C)\}$, $R_{1,B} = \{(7, B), (1, B)\}$, $p_0(t) := t.a$ modulo 2, $p_1(t) := t.b$

$\sigma_{b='C'}(R)$: selects all tuples where attribute b equals 'C' $\Rightarrow \sigma_{b='C'}(R) = \sigma_{b='C'}(R_{0,C})$

$\sigma_{a=7}(R)$: selects all tuples where attribute a equals 7 $\Rightarrow \sigma_{a=7}(R) = \sigma_{a=7}(R_{1,B})$.

$\sigma_{a=8}(R)$: selects all tuples where attribute a equals 8 $\Rightarrow \sigma_{a=8}(R) = \sigma_{a=8}(R_{0,A} \cup R_{0,C})$.

Logical/Conceptual vs Physical Index

A Logical/Conceptual Index

A logical (or conceptual) index **provides or combines concept(s)** allowing us to select a suitable subset of the horizontal partitions.

Example:
Let $P := R.a = c$ be an equality predicate.
$p_0 : [R] \rightarrow int, p_0(t) := t.a$ modulo m

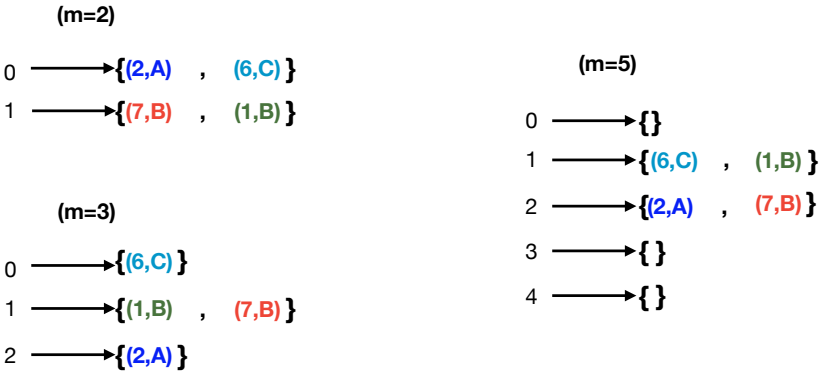
Logical Index:
1. call $p_0(c)$ to identify the qualifying horizontal partitions,
2. only inspect the partitions found for results.

A (more) Physical Index

A (more) physical index **specifies at least one of the concepts of a logical index**.

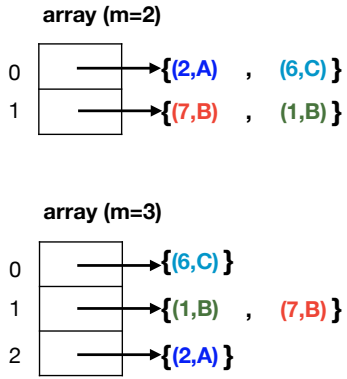
A (more) Physical Index:
1. use a **Python list type** for the different horizontal partitions, each object in that list is another **Python list** for the qualifying tuples, i.e. `list` of `list`. Everything is kept in main memory at all times.
2. use a for-loop over all tuples found in a slot. Everything is kept in main memory at all times.

Example: A Logical Index



Each of the return values of p_0 is mapped to a set of tuples (the horizontal partitions). Each mapping maps from a partition identifier to a (conceptual) set of (conceptual) tuples. How to translate the conceptual parts is the task of a (more) physical index.

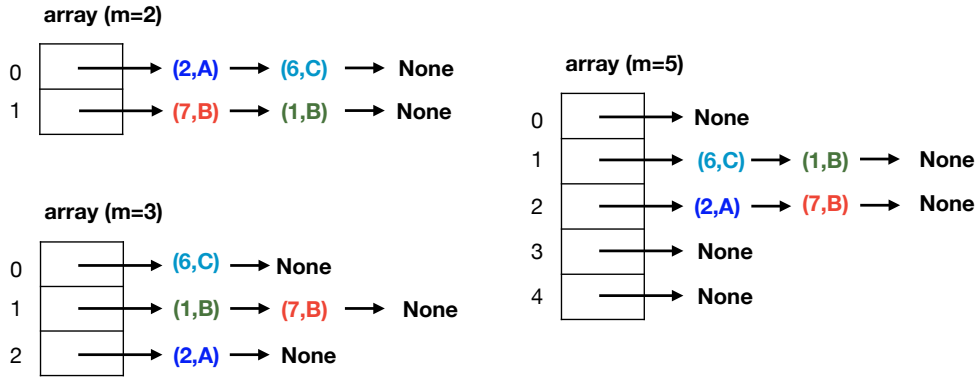
A (more) Physical Index



Each of the slots of a (conceptual) array contains a (conceptual) set of (conceptual) tuples. How to translate the conceptual parts is the task of a (more) physical index.

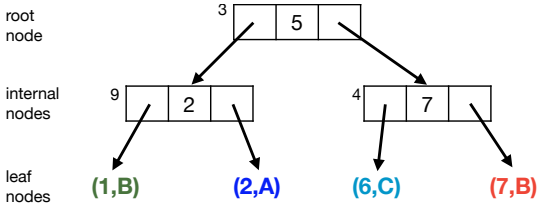
This type of index is called a *hash index*. It is one of the most important ideas in computer science. It exists in many different variants and has **z**illions of applications.

An (even more) Physical Hash Index Example



Here, each horizontal partition is implemented using a pointer structure: the array cell points to the first element in a list. 'None' ends a list.

Logical Tree Index Example



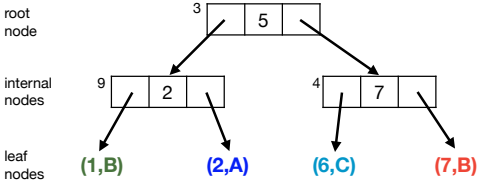
- The boxes are called *nodes*.
- The top-most node is called the root(-node).
- The internal nodes have have three entries: (left child, pivot, right child).
- Each leaf is a tuple (or a set of tuples).

Search algorithm for a constant c:

pointQuery(c, node n=root):

```
if n is an internal-node:
    if c ≥ n.pivot:
        return pointQuery(n.right, c)
    else:
        return pointQuery(n.left, c)
else:
    if n.a == c:
        return n.b
    else:
        return "not found"
```

This type of index is called a *tree index*. It is one of the most important ideas in computer science. It exists in many different variants and has **b**illions of applications.



(a) logical tree index

ID	left	pivot	right
3	9	5	4
4	(6,C)	7	(7,B)
9	(1,B)	2	(2,A)

(b) a (more) physical tree index
[representation in relational model]

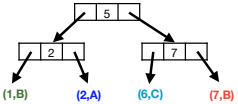
ID	left	pivot	right
3	9	5	4
4	(6,C)	7	(7,B)
9	(1,B)	2	(2,A)

(c) an (even more) physical tree index
[column layout]

001101001001
100101100011000100 10
010101110010
010001110010001000 01

(d) an (even much more) physical tree index
[binary representation]

The Physiological Index Design Continuum



(a) logical index

ID	left	pivot	right
3	9	5	4
4	(6,C)	7	(7,B)
9	(1,B)	2	(2,A)

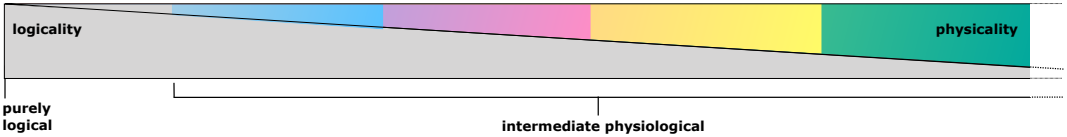
(b) a more physical index

ID	left	pivo	right
3	9	5	4
4	(6,C)	7	(7,B)
9	(1,B)	2	(2,A)

(c) an even more physical index

001101001001
100101100011000010010
010101110010
010001110010010010000 1

(d) `physical' index



This design continuum does not only exist for indexes but for many other concepts in computer science and in particular: systems and data management.

Literature: Jens Dittrich, Joris Nix: The Case for Deep Query Optimisation. Conference on Innovative Data Systems Research (CIDR). 2020.

Summary: Logical vs Physical

Take-away Message

In computer science, the terms 'logical' and 'physical' are highly misleading and often used in confusing ways. Even if something is called 'physical', there is always almost something **more physical** underneath. Vice versa, if something is called 'logical', there is always almost something **more logical** on top.

Big Data Confusion $\sim$ Physical/Logical Confusion

We already observed a similar confusion with the buzzword *big data*:

1. The word *big* is a relative term actually meaning *bigger than* X yet **without specifying the reference point** X. This makes it technically a nonsense term.
2. The word *physical* means *more physical than* Y or *increasing the physicality of* Y yet **without specifying the reference point** Y**!**
3. The word *logical* actually means *more logical than* Z or *increasing the logicality of* Z yet **without specifying the reference point** Z**!**

A file-system-based Physical Index

In [10]:
```
# print the partitioning tree created by the recursive partitioning:
# note that the leaf nodes, i.e. the csv-files are not printed
tree('myindex',' ', False)
```

```
+myindex/
  +6/
  | +5/
  |   +5/
  +1/
  | +1/
  |   +9/
  |     +7/
  +2/
    +9/
    | +0/
    | +1/
    | +4/
    | +3/
    | +2/
    | +5/
    +8/
      +9/
      +0/
      +7/
      +6/
      +1/
      +8/
      +4/
      +3/
      +2/
      +5/
```

see Notebook: Indexing by Recursive External Partitioning.ipynb