Physical design

Knowledge objectives

1. Enumerate the main basic tasks in the physical design of a DB
2. Enumerate the main criteria we should use on making a decision about the physical design of a DB
3. Enumerate the main difficulties we would find in the physical design
4. Describe the different data structures (i.e., B+, Hash, and Clustered index)
5. Describe the different access paths given a structure or lack of it (i.e., table scan, one tuple, several tuples)
6. Explain the difference in the cost of a modification and that of a query

Understanding objectives

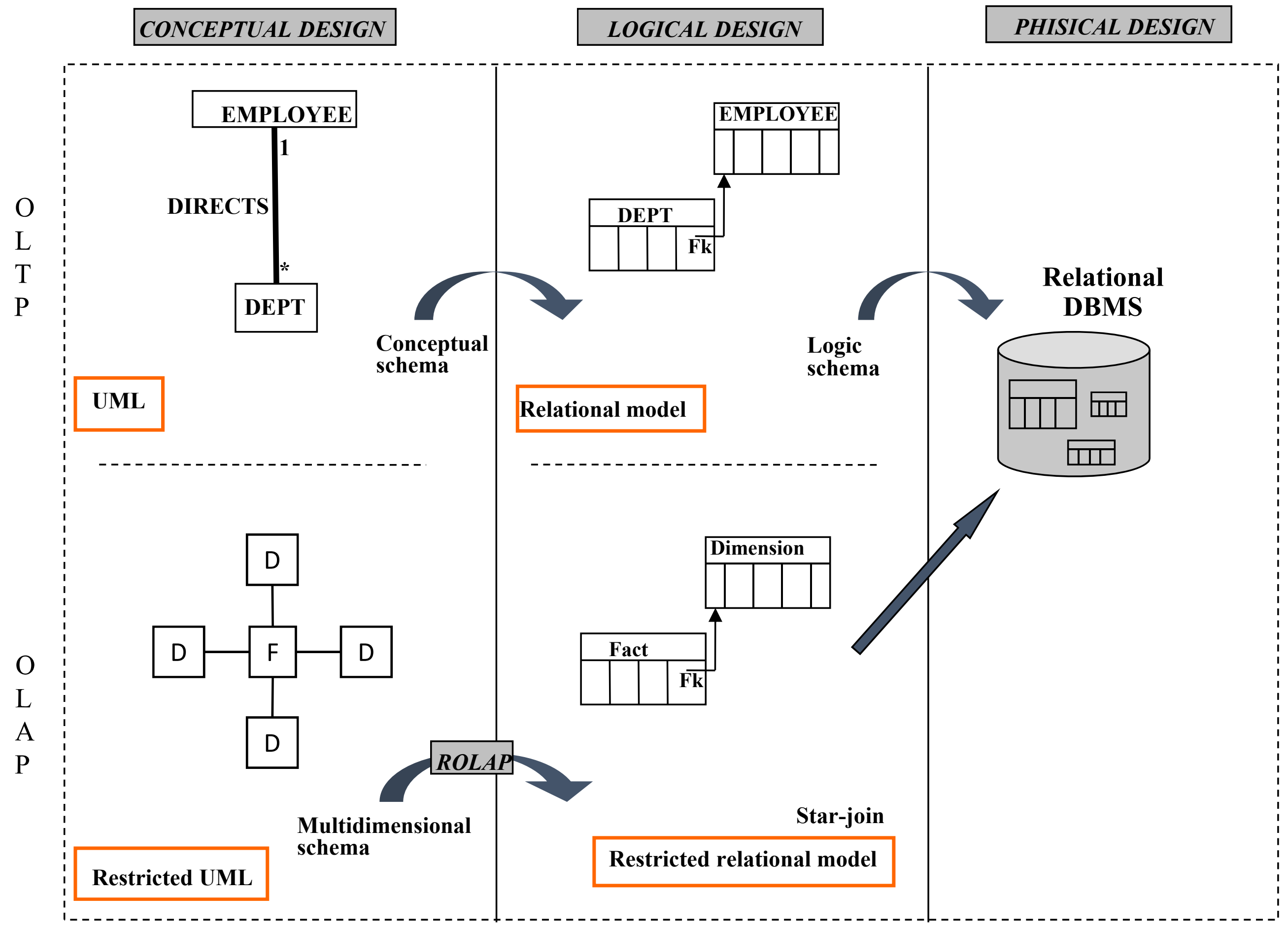
1. Use a simplified version of cost formulas to obtain an estimation of a selection when the number of selected tuples in known
2. Decide for each structure (i.e., B+, Hash, Clustered index) and operation (i.e., table scan, select one tuple, select several tuples and join), whether it is generally worth to use the structure to perform the operation and how it would be used

Tasks, criteria and difficulties

“In theory, there is no difference between theory and practice. In practice, there is.”

Jan L. A. Van de Snepscheut

Comparison of design steps



Basic tasks on physical design

- Adapt the logical schema to the DBMS
  - Data types
  - Views
  - Integrity constraints
  - Deadlocks
- Revisit the relational schema
  - Partitioning
- Choose data structures
  - Indexing
  - Materializing views
- Performance test
  - Concurrency control
  - Recovery
  - Files
  - System parameters

Criteria for physical design

- Performance improvement
  - Memory and disk space
  - CPU time
  - <u>Disk access time</u>
  - Contention
  - Auxiliary processes costs
- Scalability
- Availability
- Integrity
- Administration simplicity

Difficulties in physical design

- Users
- Opposing criteria
- Limited resources
- Imperfections in the DBMS (query optimizer)
- Communications (network)

Catalog

Catalog (or dictionary)

- It is the set of tables containing the information the system itself needs to work
  - Its contents are metadata (data regarding data)
- It is useful to manage and fine-tune the system
  - Helps to understand what the system is doing and why
- Its structure and contents differ from one DBMS to another, but there are standardized views

“Static” contents of the catalog

- System parameters
    - Size of buffers pool
    - Page size
- Table information
    - Table name, file name and kind of file
    - Name and type of each attribute
    - Indexes defined in the table
    - Integrity constraints
- Index information
    - Name and structure
    - Indexed attributes
- View information
    - Name and definition
- User information
    - Permissions

“Dynamic” contents of the catalog

- Tables
  - Cardinality: Number of tuples
  - Size: Number of blocks
- Indexes
  - Cardinality: Number of different values
  - Size: Number of blocks
  - Depth: Number of non-root levels
  - Range: Minimum and maximum existing values
- Users
  - Connection attempts

SQL’03 catalog views (I)

- DEFINITION_SCHEMA
  - Contains 61 “tables”
  - Used by the administrator

- INFORMATION_SCHEMA
  - Contains 71+43 views
    - Most of them coincide with DEFINITION_SCHEMA
  - Used by users

SQL'03 catalog views (II)

- ASSERTIONS
- ATTRIBUTES
- CHECK_CONSTRAINTS
- COLUMNS
- REFERENTIAL_CONSTRAINTS
- ROUTINES
- SEQUENCES
- TABLE_PRIVILEGES
- TABLES
- TRIGGERS
- VIEWS
- …

Access structures

B-tree index

Clustered index

Hash index

To improve query performance …

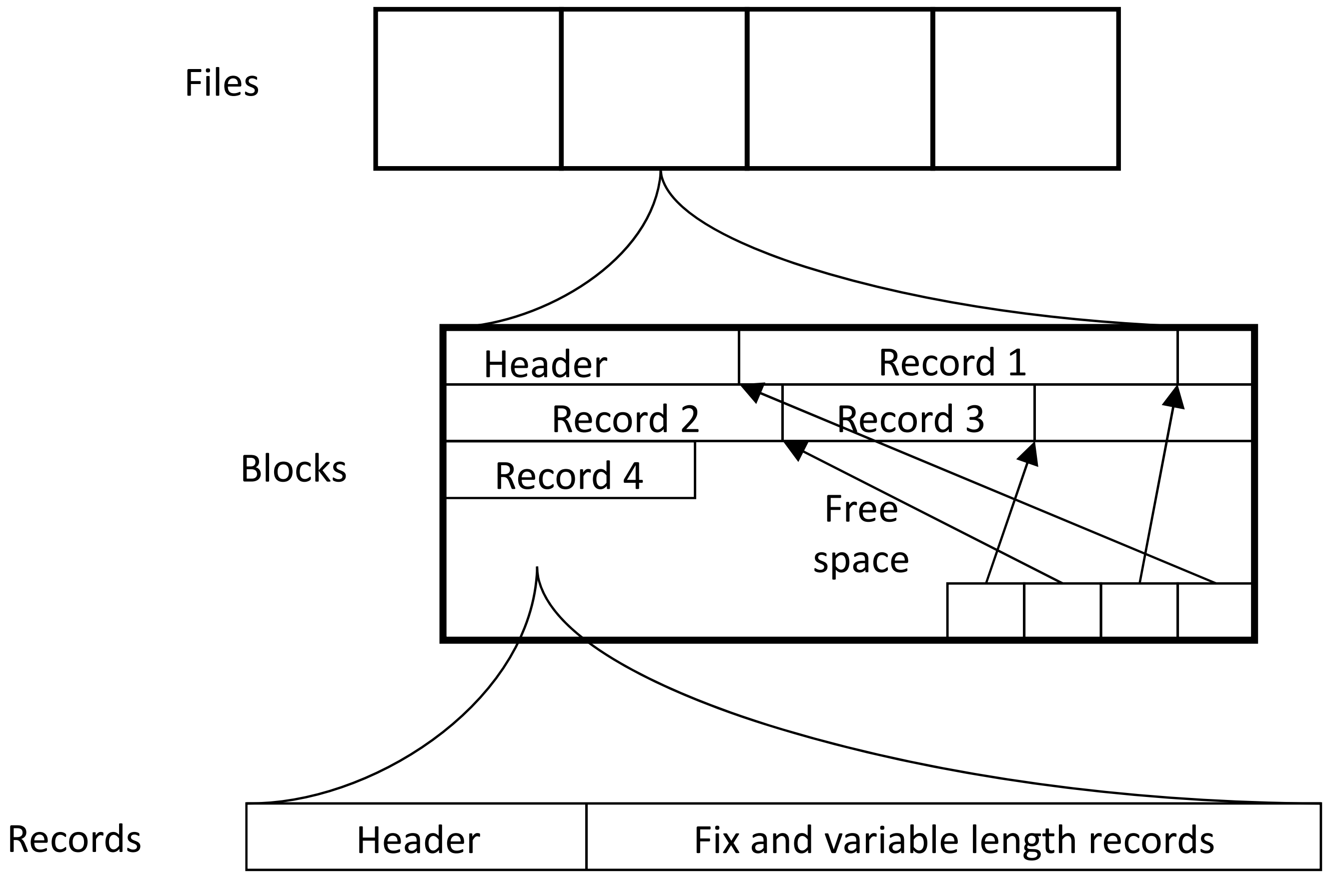
a) … Pre-compute as much as possible
- Redundant “tables” (a.k.a. materialized views)
    - Less attributes
    - Less tuples
        - Only those fulfilling the query predicate
        - Only one per combination of values of attributes in the GROUP BY
    - Less space than the table
        - Less I/O to be accessed

b) … Build access structures
- Complementary to tables
    - Less attributes
    - Same number of entries as tuples
    - Less space than the table (in general)
        - Less I/O to be accessed
            - Useful for selectivity factors next to zero (very selective)
                - Allow random access

Alternatives in the structures

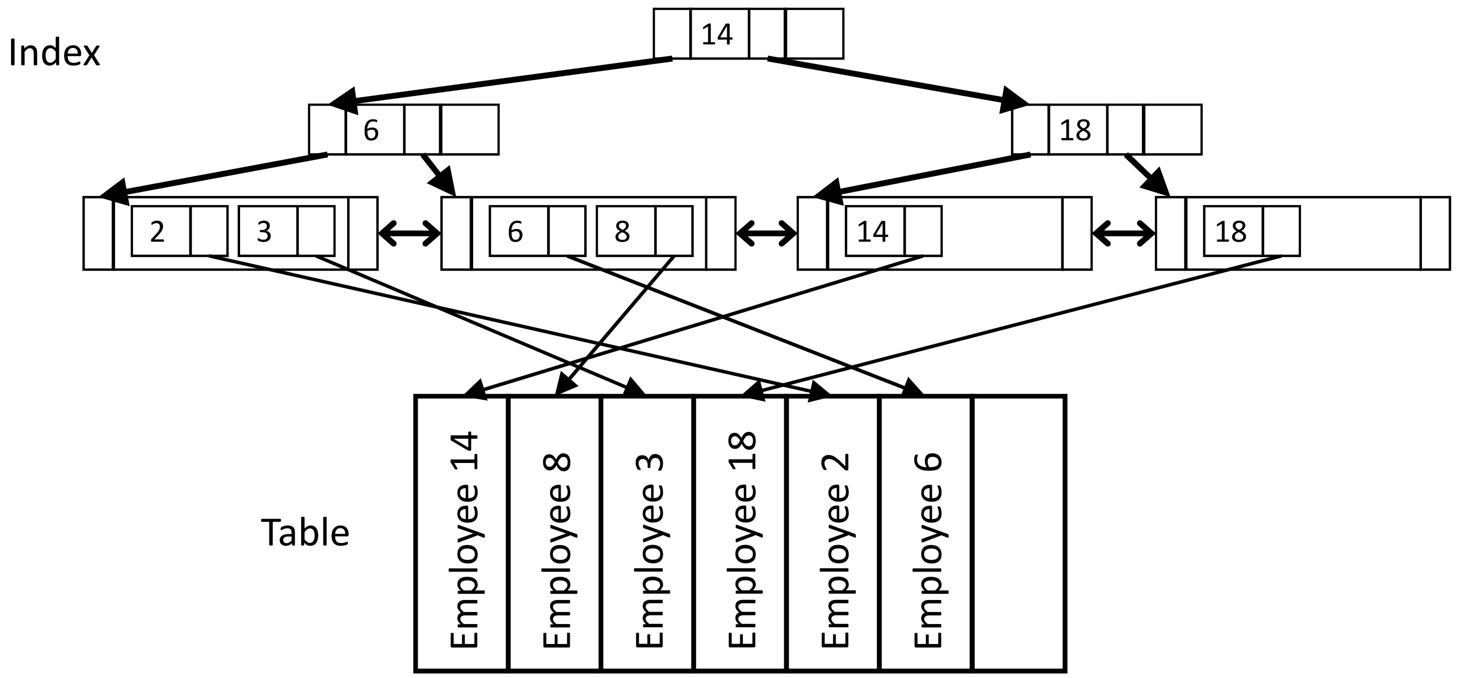
- Whether indexed or not, the table can be:
  - Ordered
  - Unordered
- Two main indexing structures:
  - B-tree
  - Hash
- Indexes always keep entries composed by pairs value-information, where information can be:
  - The whole record
  - Physical address of the record
  - List of physical addresses of records
  - Bitmap
- Null values in the indexes can be:
  - Included
  - Excluded
- Out of all possible combinations, we will only consider:
  - No index
  - Unordered and B-tree with addresses excluding NULLs (B+)
  - Ordered and B-tree with addresses excluding NULLs (Clustered)
  - Unordered and hash with addresses excluding NULLs (Hash)

Table files


Files
Blocks
Header
Record 1
Record 2
Record 3
Record 4
Free space
Records
Header
Fix and variable length records

B-tree with addresses (B+)

```
CREATE [UNIQUE] INDEX <index_name> ON <table_name> (<column_name>, …);
```

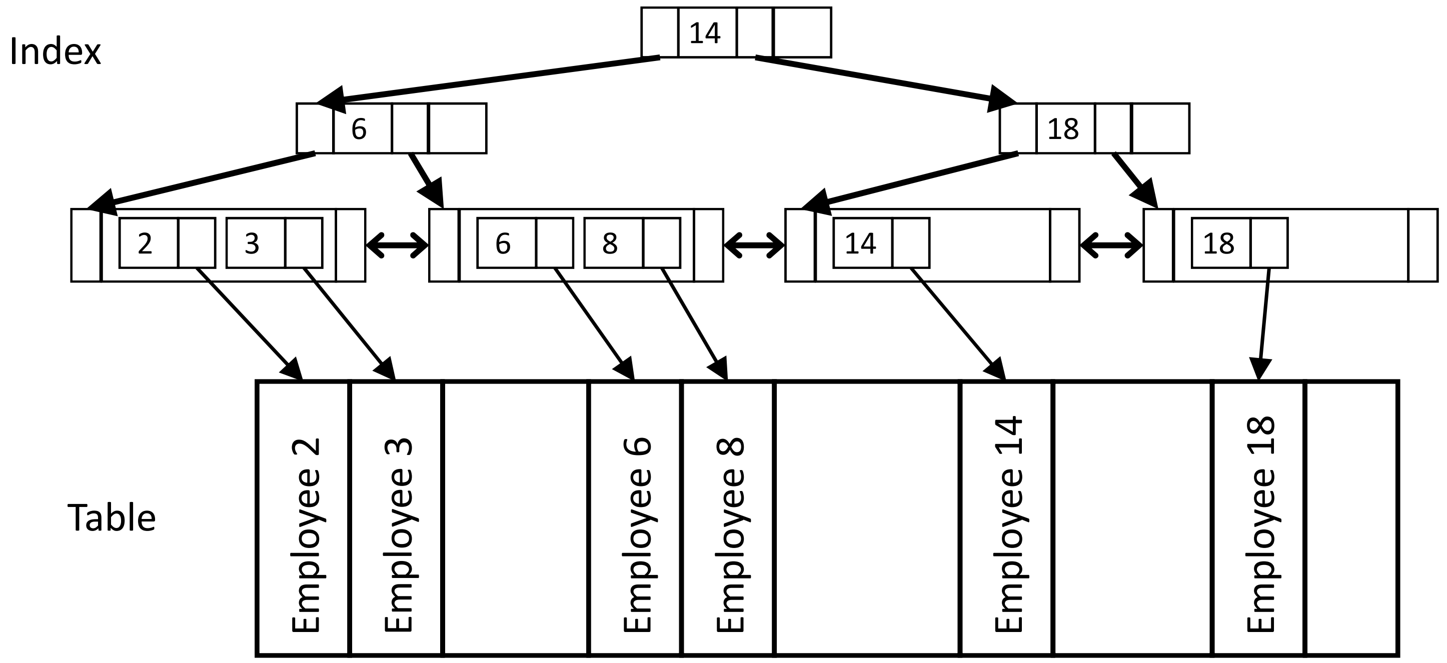


- Assumptions:
  - In every tree node (a disk block) 2d addresses fit
  - Usually, the tree load is 66% (2/3 of its capacity)

https://www.postgresql.org/docs/current/sql-createindex.html

Ordered table & B-tree with addresses (Clustered)

```
CREATE [UNIQUE] INDEX <index_name> ON <table_name> (<column_name>, …);
CLUSTER <table_name> USING <index_name>;
```

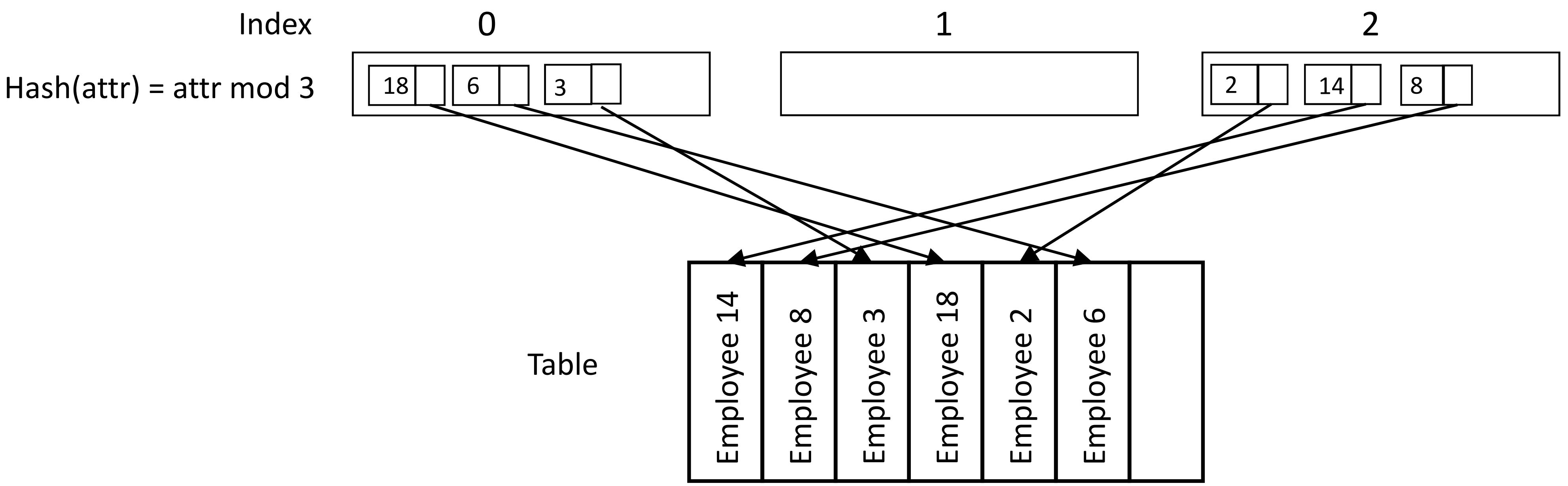


- Assumptions:
  - Only 2/3 are used in every block (for index as well as for table blocks)

https://www.postgresql.org/docs/current/sql-cluster.html

Hash with addresses (Hash)

```
CREATE INDEX <index_name> ON <table_name> USING HASH (<column_name>, …);
```



- Assumptions:
  - There are no blocks for excess
  - In a bucket block the same number of entries fit as in a tree block
  - Usually, bucket blocks are used at 80% (4/5 of their capacity)

https://www.postgresql.org/docs/current/sql-createindex.html

Size of structures

Cost variables

- B: Number of **full** blocks/pages that need the records
- R: Number of records per block/page
- D: Time to access (read or write) a disk block
  - Approximately 10^{-3} seconds (10^{-6} seconds for SSD)
- C: Time for the CPU to process a record
  - Approximately 10^{-9} seconds (we will ignore this)
- H: Time to evaluate the hash function
  - Approximately 10^{-9} seconds (we will ignore this)
- d: Tree order
  - Usually greater than 100
- |T|: Cardinality of table T

We will **not** consider the improvement due to sequential scan nor cache!

Sizes

- No index
  - B
- B+
  - $\sum_{1}^{h+1} \lceil |T|/u^i \rceil + B$
- Clustered
  - $\sum_{1}^{h+1} \lceil |T|/u^i \rceil + \lceil 1.5B \rceil$
- Hash
  - $1 + \lceil |T|/(0.8 \cdot 2d) \rceil + B$

$u = \%load \cdot 2d = (2/3) \cdot 2d$
$h = \lceil \log_u |T| \rceil - 1$

Example of nodes and height in a B+

$|T| = 64$

$d = 3$

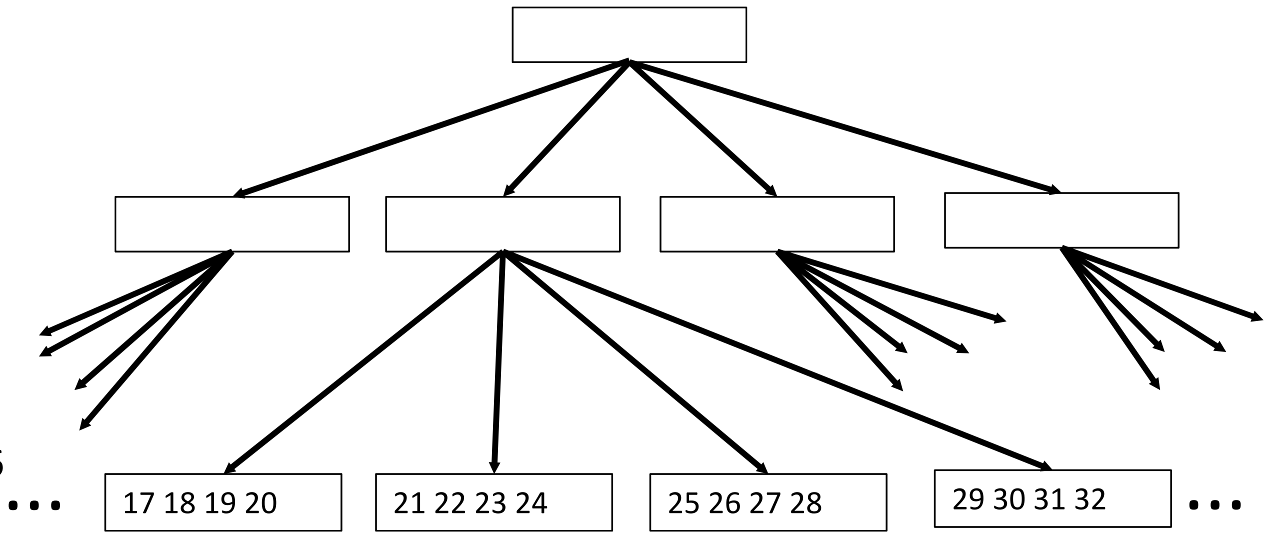
$\%load = 2/3$

$u = \%load \cdot 2d = 4$ (entries per node)

$h = \lceil \log_u |T| \rceil - 1 = \lceil \log_4 64 \rceil - 1 = 2$

#nodes: $\sum \lceil |T|/u_i \rceil = 16 + 4 + 1 = 21$

LEVEL:	Nodes in level
3:	$64 / 4^3 = 1$
2:	$64 / 4^2 = 4$
1:	$64 / 4^1 = 16$



Example of nodes and buckets in a Hash

|T| = 64

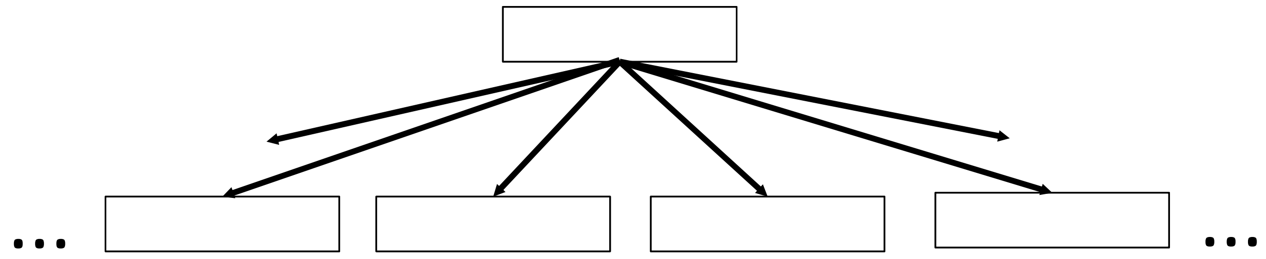
Full capacity = 6 (assume the same as a B+)

%load = 4/5 = 0.8

#buckets = $\lceil |T|/(0.8 \cdot 2d) \rceil = \lceil 64/(0.8 \cdot 6) \rceil = 14$

#nodes = 1 + 14

LEVEL:	Nodes in level
Root:	1
Buckets:	14



Access costs

Access paths

- Table Scan

- Search one tuple
  - Equality of unique attribute

- Search several tuples
  - Interval
  - Not unique attribute

- Insertion of one record

- Deletion of one record

Table Scan

$u = \%load \cdot 2d = (2/3) \cdot 2d$

- No index
  - B
- B+
  - $\lceil |T|/u \rceil + |T|$ **Only useful to sort**
- Clustered
  - $\lceil 1.5B \rceil$
- Hash
  - $\lceil |T|/(0.8 \cdot 2d) \rceil + |T|$ **Only useful to group**

Select one tuple

$u = \%load \cdot 2d = (2/3) \cdot 2d$
$h = \lceil \log_u |T| \rceil - 1$

- No index
  - 0.5B
- B+
  - h + 1
- Clustered
  - h + 1
- Hash
  - 1 + 1

Select several tuples

- No index
  - B
- B+
  - $h + (|O|-1)/u + |O|$
- Clustered
  - $h + 1 + (|O|-1)/((2/3)\cdot R)$
- Hash
  - $v=1$: $1 + k$
  - $v>1$: $v\cdot(1 + k)$
  - v is unknown: Useless

$u = \%load\cdot 2d = (2/3)\cdot 2d$
$h = \lceil \log_u |T| \rceil - 1$
O = output table
v = values in the range
k = repetitions per value
$|O| = v * k$ (usually computed by other means)

Insertion of one tuple

$u = \%load \cdot 2d = (2/3) \cdot 2d$
$h = \lceil \log_u |T| \rceil - 1$

- No index
  - 1 + 1
- B+
  - h + 1 + 1 + 1 (maybe more if split is needed)
- Clustered
  - h + 1 + 1 + 1 (maybe more if split or displacement are needed)
- Hash
  - 1 + 1 + 1 + 1

Deletion of <u>one</u> tuple

$u = \%load \cdot 2d = (2/3) \cdot 2d$
$h = \lceil \log_u |T| \rceil - 1$

- No index
  - 0.5B + 1
- B+
  - h + 1 + 1 + 1   (maybe more if merge is needed)
- Clustered
  - h + 1 + 1 + 1   (maybe more if merge is needed)
- Hash
  - 1 + 1 + 1 + 1

Index choice

Considerations on indexes

- Specially useful for simple queries
  - Without grouping, aggregations, or many joins
- Work better for very selective attributes (i.e., few repetitions per value)
  - Attributes in multidimensional queries are usually not very selective
- We can define as many indexes as we want
  - We can define only one Clustered index
  - For big tables, they may use too much space

We should define an index if …

- B-tree:
  - There is a very selective condition
- Hash:
  - There is a very selective condition (with equality)
  - The table is not very volatile
  - The table is huge
- Clustered:
  - There is a little selective condition, or a GROUP BY or an ORDER BY or …
  - The table is not very volatile

We should NOT define an index if …

- Processing is massive
  - Never one tuple at a time
- The table has few blocks
- The attribute has few values
  - Little selective conditions
- The attribute appears in the predicate inside a function
  - Though the DBMS may allow function-based indexes

Index-only query answering

Beyond that, indexes can be used to completely avoid the access to the table:

- Projection

```
SELECT age                    SELECT DISTINCT age
FROM people;                  FROM people;
```

- Aggregates

```
SELECT MIN(age)               SELECT AVG(age)
FROM people;                  FROM people;

              SELECT age, COUNT(*)
              FROM people
              GROUP BY age;
```

- Joins

```
SELECT p.name
FROM people p, departaments d
WHERE p.id=d.boss;
```

  - Index Sort-Match Join
  - Index Block Nested Loops
  - Index Row Nested Loops

Closing

Summary

- Catalog
- Structures
  - B+
  - Hash
  - Clustered index
- Access paths
  - Table scan
  - Select one tuple
  - Select several tuples
  - Insert one tuple
  - Delete one tuple
- Index choice
  - Index-only query answering

Bibliography

- Y. Ioannidis. *Query Optimization*. ACM Computing Surveys, vol. 28, num. 1, March 1996
- G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases*. Addison-Wesley, 1998
- J. Sistac. *Sistemes de Gestió de Bases de Dades*. Editorial UOC, 2002.
- R. Ramakrishnan and J. Gehrke. *Database Management Systems.* McGraw-Hill, 3rd Edition, 2003
- J. Lewis. *Cost-Based Oracle Fundamentals*. Apress, 2006
- S. Lightstone, T. Teorey and T. Nadeau. *Physical Database Design*. Morgan Kaufmann, 2007