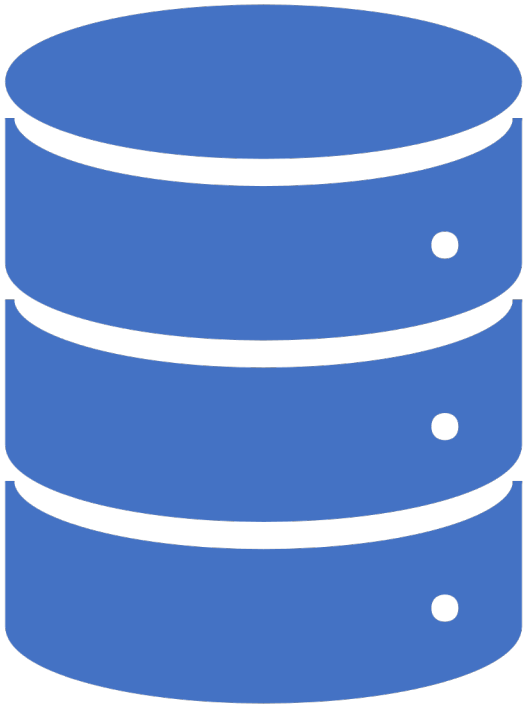


Lecture 9

Indexes

Indexes

Indexes = auxiliary data structures / files that speeds up the operations that cannot be efficiently performed relative to the file's organization, by accelerating the execution of the queries

- enable the retrieval of the rid (record id) of the records that fulfill the selection condition

e.g. Display the students that have the *age* = 21

Display the students that have the *grade* between 8 and 10

- The sequential passing through the file is expensive

e.g. The file of students that are sorted by their names

- good organization: return the students alphabetically ordered
- bad organization: retrieve the students that have the age in a given range; retrieve the students from Cluj Napoca

- When the data is ordered in the file
  - binary search to find the first record (expensive)
  - the file has to be scanned to find the other records

Indexes

- A index is created based o a *search key*
- **Search key** = set of one or more attributes of the indexed file (different than the *key* that identify the record) used to search the records from a table / file
- Search key is different than the primary key / candidate key / super-key
- The index speeds up the queries with equality or with range selection condition on the search key
- An index contains a collection of special records and allows the finding of all the records from the indexed table for which the search key has the value given (k).
- **Entries** = the records from the index (e.g. (search_key, rid)) that are enabling the retrieval of records with a given search key value

Indexes - example

- files with the records of the students
- the index is created on the attribute *age*
- entries: (age, rid), with rid that identifies a student record
- this index will increase / speed up the queries about the students that have the age given
- age: find the entries in the index with age=21; follow rid from the obtained entries to retrieve the records that describe the students with the age 21

Indexes

- An index can improve the efficiency of certain types of queries, but not all of them (improve the queries that use the attributes on which are created indexes)
- The organization techniques / access methods: B+ trees, hash-based structures
- The index also contain information to manage the search to the required records
- Index size: as small as possible because are used in the main memory for the searches
- If the index is too *big*
  - use the partial indexing structure
  - index the indexed file (layered on a tree structure)
- By changing the data in the file is involved the update of the indexes associated with the file (e.g. insert records, update search key columns, update columns that are not part of the key, but are in the index)
  - insert / delete records from the indexed table involves the update of all the indexes defined on that table
  - update the values of the fields that are part of the search key means the update of the indexes based on that search key

- What is the contain of an index (data entry)?
- How are the entries of the index organized?

Indexes

Let ***k*** be a data entry in an index. The data entry:

1. is an actual / original data record with the search key (=k)
2. is the pair <k, rid>, where rid is the id of a data record with the search key value=k
3. is the pair <k, rid_list>, where rid_list is the list of ids of data records with the search key value=k

Option 1

- The index and the original records are stored together (the file of data records does not need to be stored in addition to the index)
- The index is seen as a special file organization
- only one index / collection of records should be created (otherwise, duplicate records → redundancy → possible inconsistencies)
- If the records are large, the number of the blocks in which are stored has to be also large → dimension of the auxiliary information from the index is also large

Indexes

Option 2, 3

- Data entries point to the corresponding data records (index entries correspond to the original records)
- In general, the size of the entry is smaller than the size of a data record (so, are more efficient than the *Option 1*, especially because the search key is smaller)
- The auxiliary information from the index used to direct the search, it is also reduced than the one from *Option 1*
- The *Option 3* is more compact that *Option 2*, but can contain variable-length records (because, involves the variable dimension of the store space of the entries even if the search key has a fixed dimension)
- Can be used by several indexes on a collection of records
- Independent of the file organization

Indexes

Indexes – Data Entries – Create an index

search key: Age
data entries: <search_key, rid>

	StudentName	Age	Grade
rid 1	Nicholas	20	9.50
rid 2	Manuel	21	8.75
…	…		
rid n	Carina	20	10.00

search key

rid 1: 20
rid 2: 21
…
rid n: 20

20: rid 1, rid n
21: rid 2
…

…
20: rid 1, rid n
21: rid 2
…

data entry (ordered)

index file (switched)

Indexes

- Data records – referes to the records
- Data entries – refers to the indexes

Index types:

- **primary index** – the search key includes the primary key
  - cannot contain duplicates
- **secondary index** – the search key does not include the primary key
  - can contain duplicates
- **unique index** – the search key contains a candidate key
  - cannot contain duplicates (for the data entries)
- duplicates – data entries with the same search key value

Indexes

Index types:

- **clustered index** – (grouped) – the order of the data records is close to / the same as the order of the data entries (in the index)
  - *Option 1* involves grouping – should be used
  - A table can be indexed by only one key
  - The cost to find the data by using the index is influenced by the grouping performed.
  - To construct grouped indexes
    - the records are ordered inside of the file (*heap file*)
    - in each memory page is reserved space for the future inserts
      - if the extra space is used, will be formed linked lists of supplementary pages
      - a regular organization is necessarily to ensure a high performance
    - the maintenance of the grouped indexes is expensive

- **unclustered index** – index that is not clustered

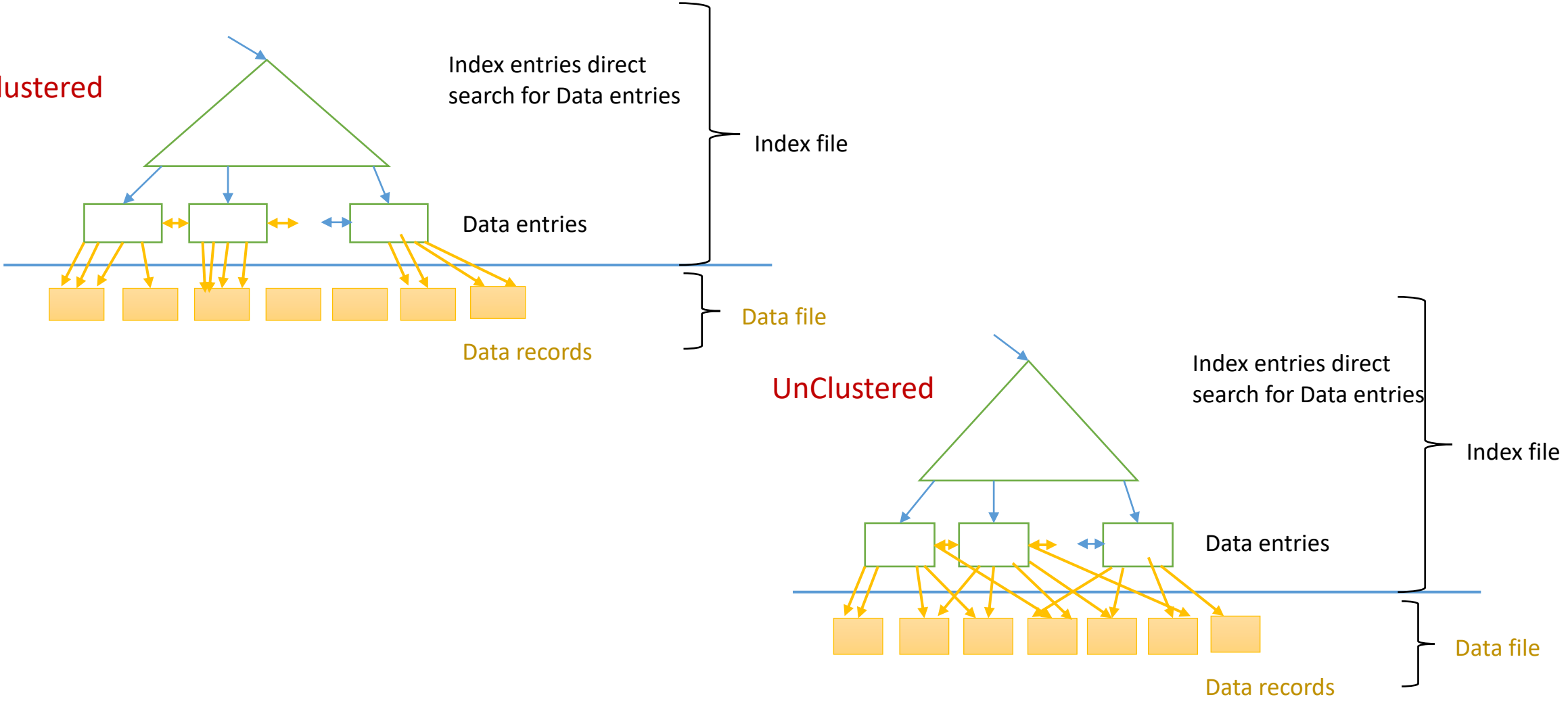
Indexes

Index types: **clustered index / unclustered index**

- index that use *Option 1* – clustered (from the definition, because the data entries are actually data records)
- index that use *Option 2, 3* – clustered only if the data records are ordered by the search key
- in practice:
  - expensive to maintain the sort order for files (rarely kept sorted)
  - clustered index = index that use *Option 1*
  - unclustered index = index that use *Option 2, 3*
- On a collection of records: ***at most one clustered index, several unclustered indexes***
- Range search key (e.g. *Age between 20 and 26*) - cost of using an unclustered index
  - each data entry that satisfies the condition in the query could contain a rid pointing to a distinct page
  - the number of I/O operations could be equal to the number of data entries that satisfy the query's condition

Indexes

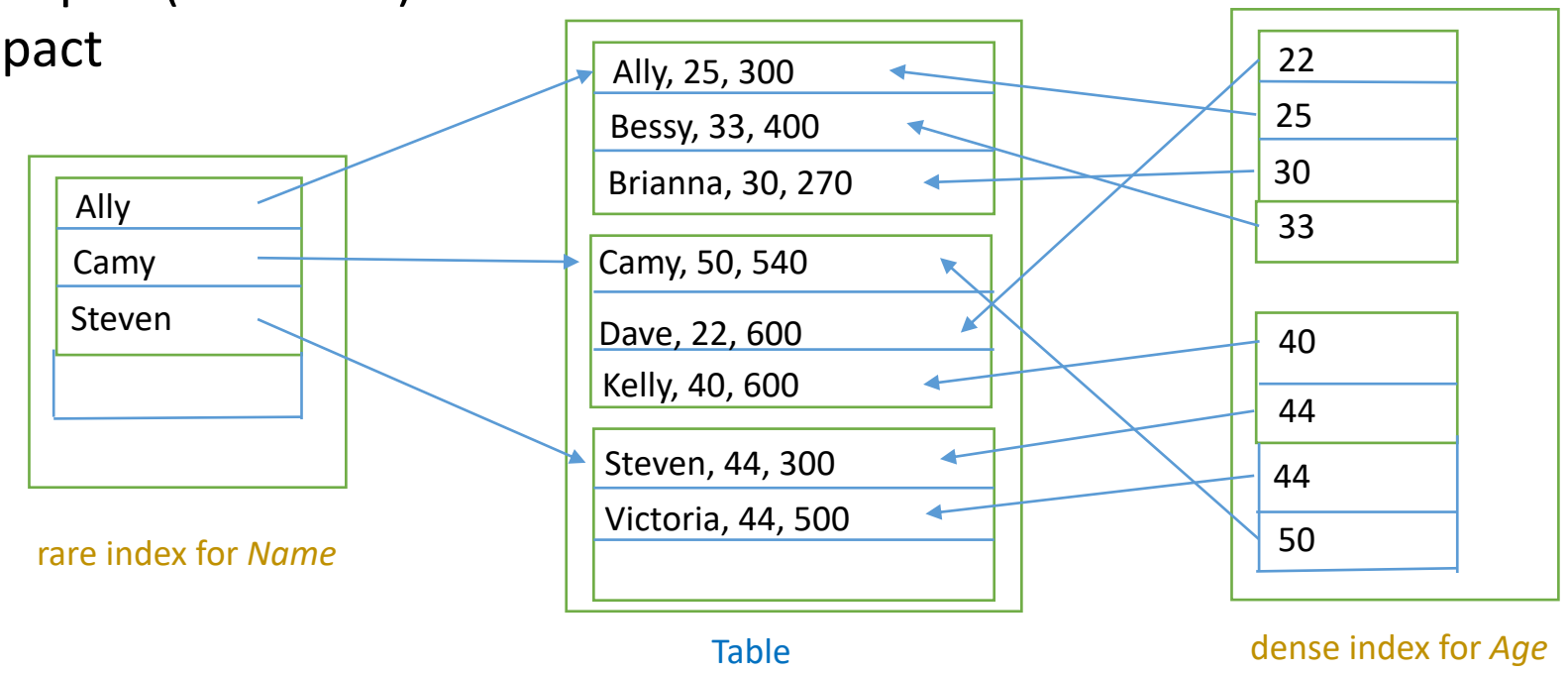
Index types: **clustered index / unclustered index**



Indexes

Index types: dense / rare indexes

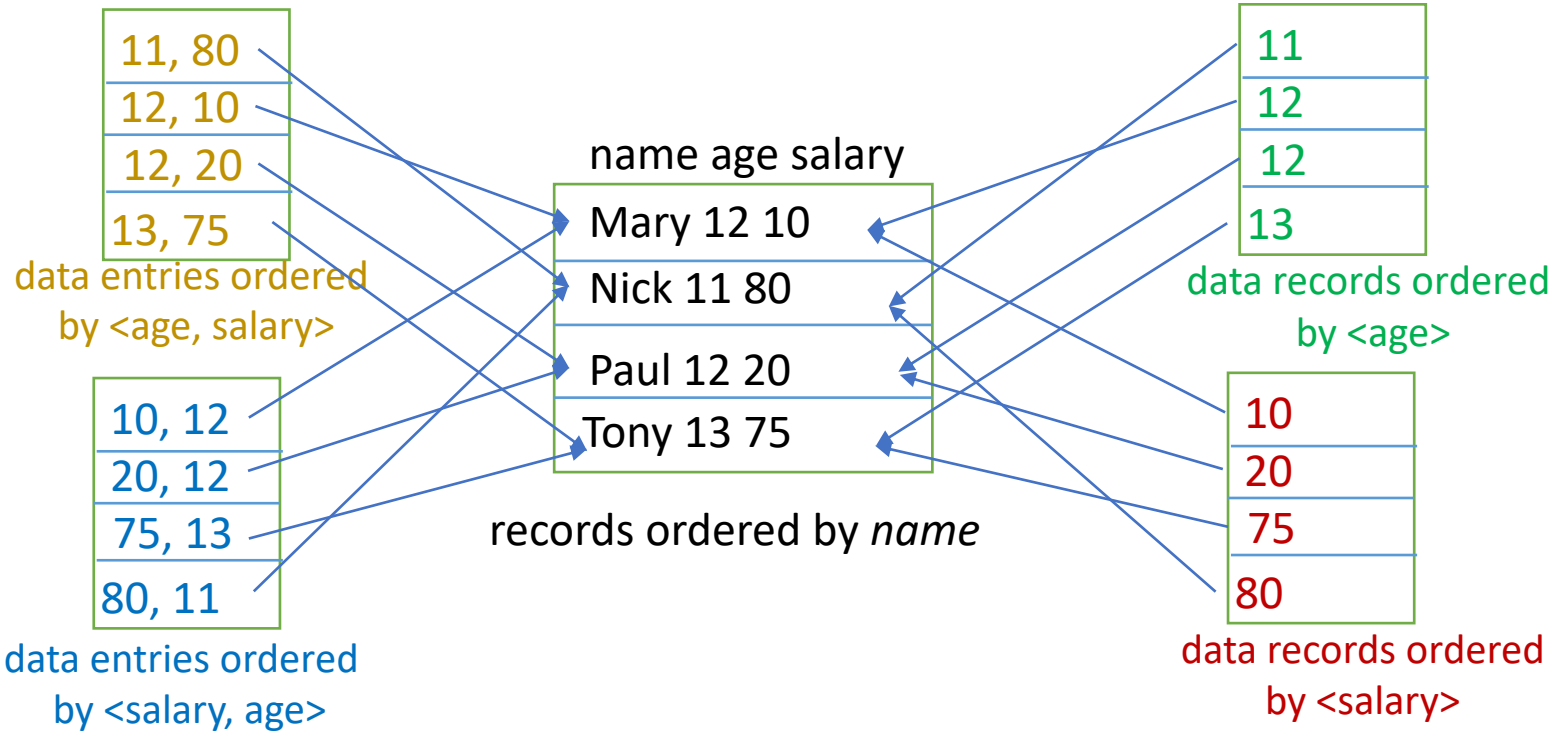
- **dense index** = the index that has at least one data entry for each value of the search key (that can be found in the data records)
  - Multiple data entries can have the same value for the search key – *Option 1*
  - *Option 1* – assure that the index in dense
- **rare index** = the index that memorize a data entry for each record page
  - All the rare indexes are grouped (clustered)
  - The rare indexes are compact



Indexes

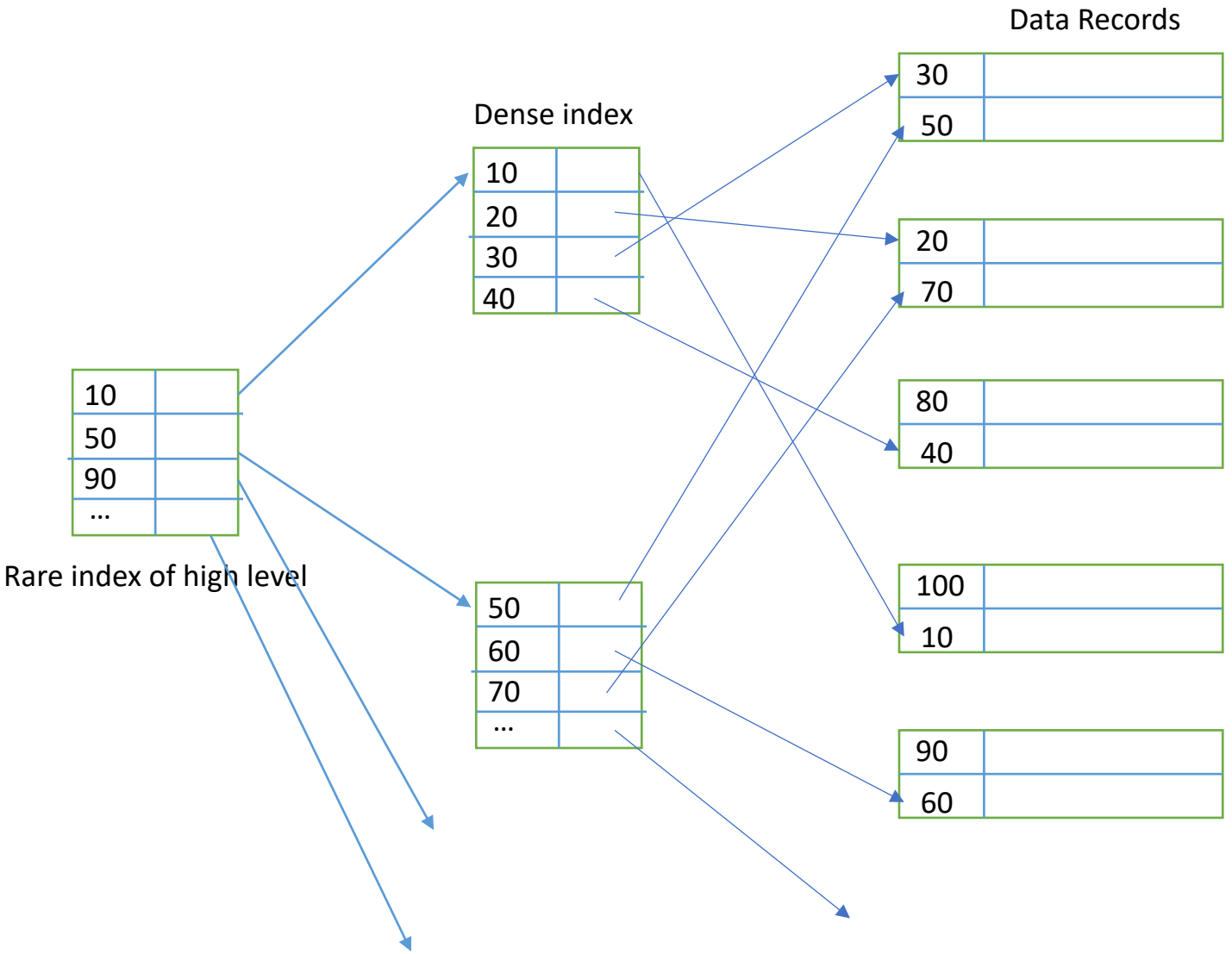
Composite search key

- **composite (concatenated) search key** = search key that contains several fields
- useful when the search is done on a combination of fields
  - equality queries (the value of a field is equal with a constant) (e.g. *age=21 and salary=500*)
  - interval queries (the value of a field is not a constant) (e.g. *age =21 and salary >500)*
- data entries (indexes) are ordered by the search key to be useful to the interval queries
  - alphabetically ordered
  - spatial ordered



Indexes

Multilevel indexes



Indexes

Example: The records of the table *Student* are stored in a file and are ordered by *Age*. Each page may store a maximum number of 3 records. How many data entries exist (use <page_id, slot_no> to identify a tuple)?

StudentId	StudentName	Age	Grade	Course
12	Mary	11	7.8	Biology
13	John	12	8.0	Biology
54	Paul	18	9.4	Chemistry
23	Nick	19	9.2	Biology
48	Nick	19	9.8	Chemistry

- *Age* – all the table (dense index) – *Option 1*
- *Age* – (11, <1,1>), (12, <1,2>), (18, <1,3>), (19, <2,1>), (19, <2,2>) (dense index) – *Option 2*
- *Age* – (11, <1,1>), (12, <1,2>), (18, <1,3>), (19, <2,1>, <2,2>) (dense index) – *Option 3*
- *Age* – such index cannot be built (rare index) – *Option 1*
- *Age* – (11, <1,1>), (19, <2,1>) (rare index) – *Option 2* – the order of the records is important
- *Age* – (11, <1,1>), (19, <2,1>, <2,2>) (rare index) – *Option 3* – the order of the records is important

Indexes

Example: The records of the table *Student* are stored in a file and are ordered by *Grade*. Each page may store a maximum number of 3 records. How many data entries exist (use <page_id, slot_no> to identify a tuple)?

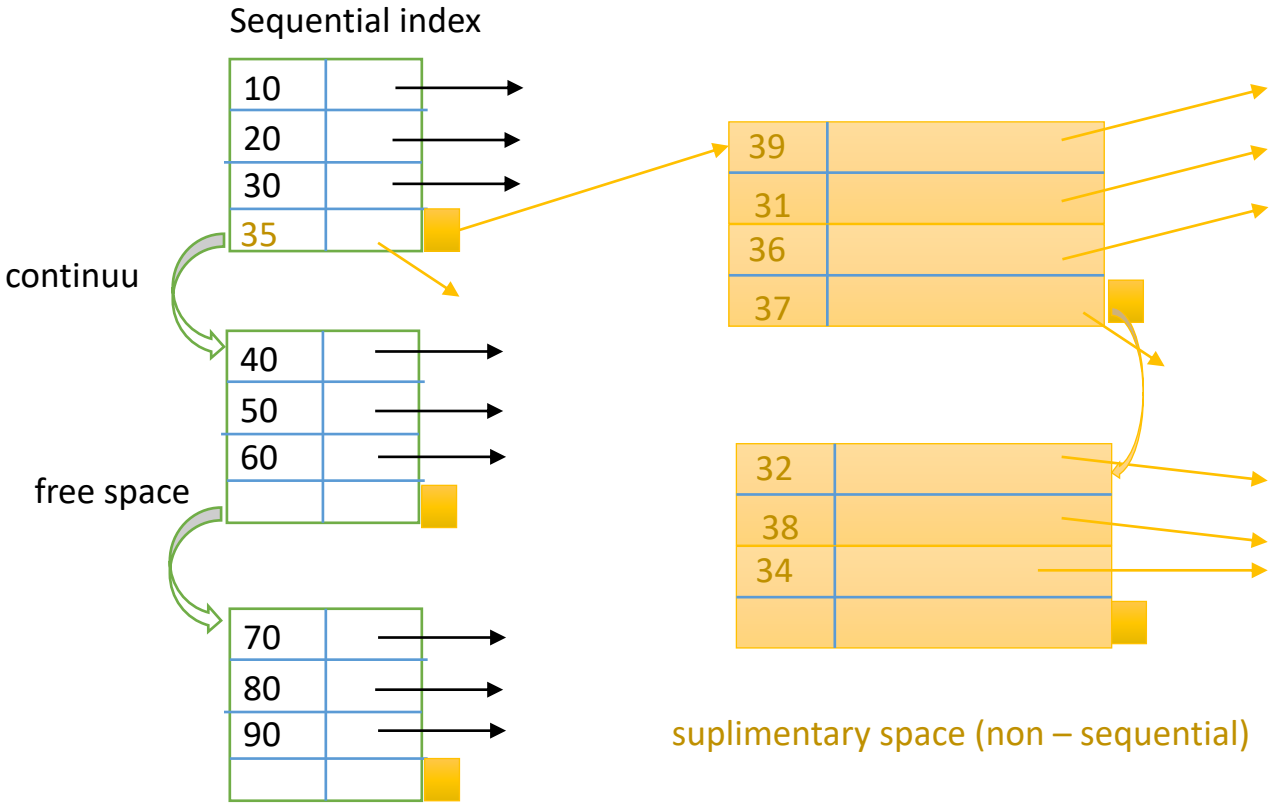
StudentId	StudentName	Age	Grade	Course
12	Mary	11	7.8	Biology
13	John	12	8.0	Biology
54	Paul	18	9.4	Chemistry
23	Nick	19	9.2	Biology
48	Nick	19	9.8	Chemistry

- *Grade* – 7.8, 8.0, 9.2, 9.4, 9.8 (dense index) – *Option 1*
- *Grade* – (7.8, <1,1>), (8.0, <1,2>), (9.2, <2,1>), (9.4, <1,3>), (9.8, <2,2>) (dense index) – *Option 2*
- *Grade* – (7.8, <1,1>), (8.0, <1,2>), (9.2, <2,1>), (9.4, <1,3>), (9.4, <2,2>) (dense index) – *Option 3*
- *Grade* – such index cannot be built – from definition (rare index) – *Option 1*
- *Grade* – it is not possible (rare index) – *Option 2* – the values of the search key are not ordered
- *Grade* – it is not possible (rare index) – *Option 3* – the values of the search key are not ordered

Indexes

Conventional indexes

- are simple; are sequential files that are suitable to cross the file
- the inserts are expensive and loose the sequential ability and the equilibrium



Indexes

***Workload* understanding**

- to each used query:
  - which are the relations / tables used?
  - which are the fields returned?
  - which are the attributes involved in the select / join conditions?
- to each modification:
  - which are the attributes involved in the select / join conditions? How selective are the conditions?
  - what type of modification is (INSERT / UPDATE / DELETE)? And which are the affected attributes?

Choosing indexes

- the most important queries are analyzed; to each one is considered the most optimal execution plan by using the existent indexes; a new index is created only if could generate a better execution plan
- before create a new index, should be evaluated the impact of its modification: indexes are increasing the execution of the queries , but slow the modifications; also, need supplementary store space
- the attributes from the WHERE clause are *candidate* search keys: the equalities suggest a structure with access direct; the intervals suggest the usage of a tree structure (the group is important here and also for the equalities where are a lot of duplicates)

Indexes

Choosing indexes

- when the WHERE clause has multiple conditions can be taken into consideration composed search keys: the order of the attributes is important to evaluate the conditions with intervals; for the important queries these indexes can determine *index-only* execution strategies
- will be chosen the indexes that are useful for multiple queries; only one grouped index can be defined on a table, and so, should be used in an important query

Composed search keys

- To return the students with *age=20* and *grade=9* an index on *<age, grade>* is more suitable than an index on *age* or an index on *grade*
- for 19<*age*<26 AND 8<=*grade*<=10 a tree structured index is good: *<age, grade>* or *<grade, age>*
- for *age=20* AND 8<=*grade*<=10 a tree structured index *<age, grade>* is better than an index *<grade, age>*
- the composed indexes are bigger and are modified frequent

Examples: SELECT StudentName FROM Student WHERE Age>19

- SELECT StudentName, COUNT(*) FROM Student WHERE Age>19 GROUP BY StudentName
  - an index on *StudentName* should be used
- SELECT StudentName FROM Student WHERE Course='Biology'
  - a grouped index on *Course* should be used

Indexes

Indexing

- one dimension indexes: B+ Trees, Hash Files
- multi dimensional indexes: to spatial databases (GIS), R-Trees
- multi dimensional indexes: queries on intervals
- advanced index: internal memory index

Index – Only Plans

- there are queries that can be executed without accessing the original tables when a suitable index is available
- SELECT S.Age FROM Student S, Exam E WHERE S.Sid=E.Sid – index on ***E.Sid***
- SELECT S.Age, E.Grade FROM Student S, Exam E WHERE S.Sid=E.Sid – index on ***<E.Sid, E.Grade>*** - tree index
- SELECT E.Sid FROM Exam E GROUP BY E.Sid – index on ***E.Sid***
- SELECT E.Sid, MIN(E.Grade) FROM Exam E GROUP BY E.Sid – index on ***<E.Sid, E.Grade>*** - tree index
- SELECT AVG(E.Grade) FROM Exam E WHERE EDate='10/10/2021' AND E.Grade BETWEEN 7 AND 10 – index on ***<E.EDate, E.Grade>*** or ***<E.Grade, E.EDate>*** - tree index
- SELECT E.Sid, COUNT(*) FROM Exam E WHERE E.Grade=9 GROUP BY E.Sid – index on ***<E.Grade, E.Sid***> or <***E.Sid, E.Grade>*** - tree index (for E.Grade>9 index ***<E.Grade, E.Sid***> is better than <***E.Sid, E.Grade>)***

References:

- C.J. Date, *An Introduction to Database Systems (8th Edition)*, Addison-Wesley, 2003.
- H. Garcia-Molina, J. Ullman, J. Widom, *Database Systems: The Complete Book, Prentice Hall Press*, 2008.
- G. Hansen, J. Hansen, *Database Management And Design (2nd Edition),* Prentice Hall, 1996.
- R. Ramakrishnan, J. Gehrke, *Database Management Systems*, McGraw- Hill, 2007. http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- R. Ramakrishnan, J. Gehrke, *Database Management Systems (2nd Edition),* McGraw-Hill, 2000.
- A. Silberschatz, H. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill, 2010. http://codex.cs.yale.edu/avi/db-book/
- L. Țâmbulea, *Curs Baze de date*, Facultatea de Matematică și Informatică, UBB, 2013-2014.
- J. Ullman, J. Widom, *A First Course in Database Systems*, http://infolab.stanford.edu/~ullman/fcdb.html