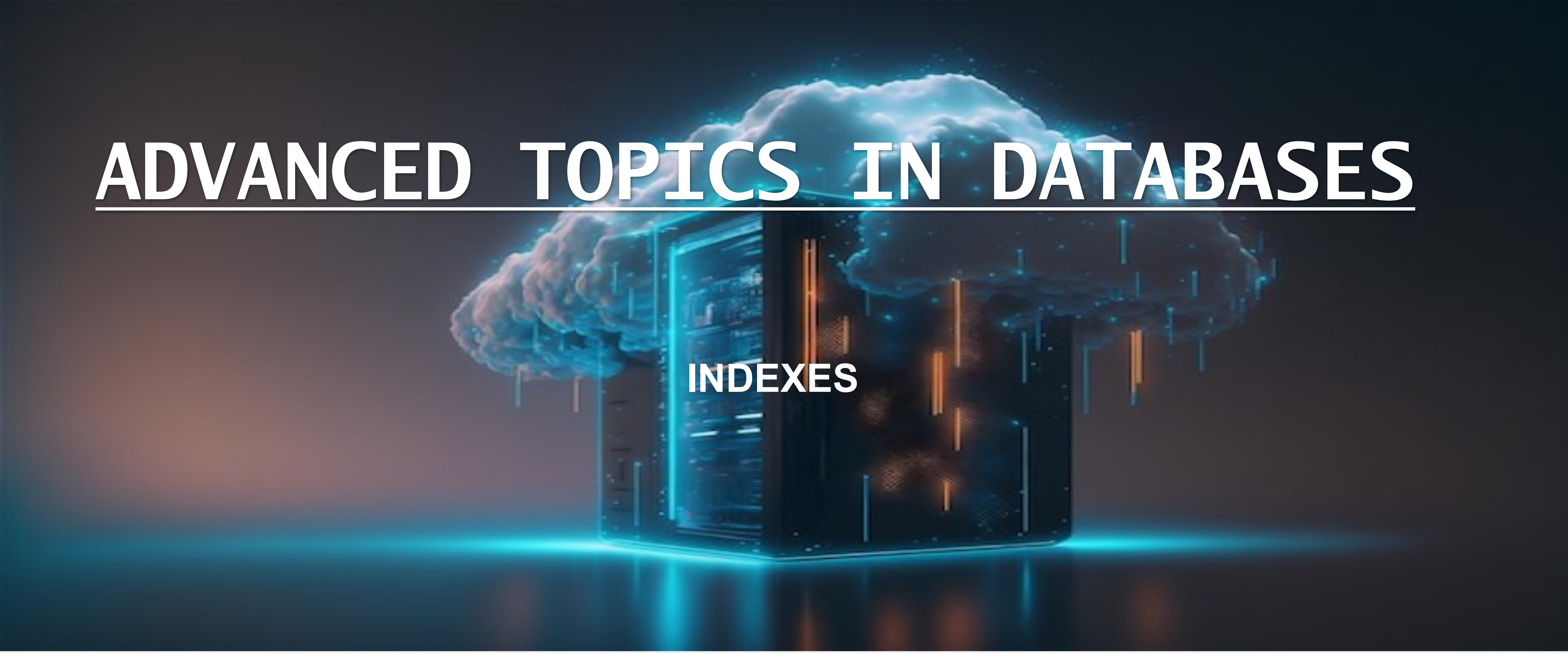


ADVANCED TOPICS IN DATABASES

INDEXES

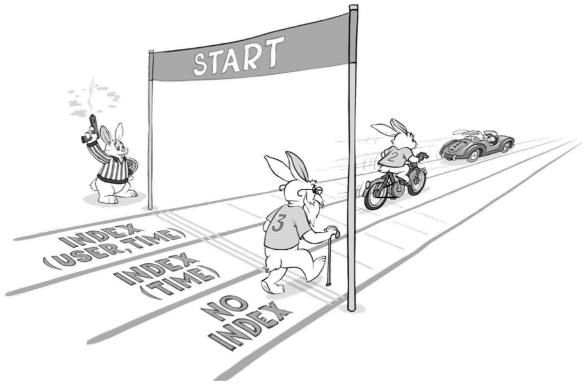
Master in Informatics Engineering
Data Engineering

Informatics Engineering Department

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

What is a Index

- **Indexes or secondary access structures** are auxiliary data structures that aim to **minimize** record **access time** in response **to search operations** on certain attributes.
- Indexes are built based on the values of one of the attributes (indexing attribute).
  - Any attribute can be used to construct an index.
  - Different attributes can be used to construct various indices.
- Indexes do not change the physical arrangement of records in disk blocks.
- **Indexes in databases are lists of values (index key) and pointers stored in a memory structure (index table), pointing to the physical location of information, in the data files associated with the tables.**



Indexes

- DBMS automatically create indexes for certain types of constraints (**PRIMARY KEY and UNIQUE**)
- The benefits of indices have costs
  - The **space allocated** in the database **is greater**
  - **Insert and delete** instructions **have longer processing time** due to the respective index maintenance operations
  - **Update instructions** may **take longer processing time due** to the respective index maintenance operations, **if the update concerns at least one of the index columns (index key)**

```
CREATE INDEX {index_name} ON
{table_name} ({column_names});
```

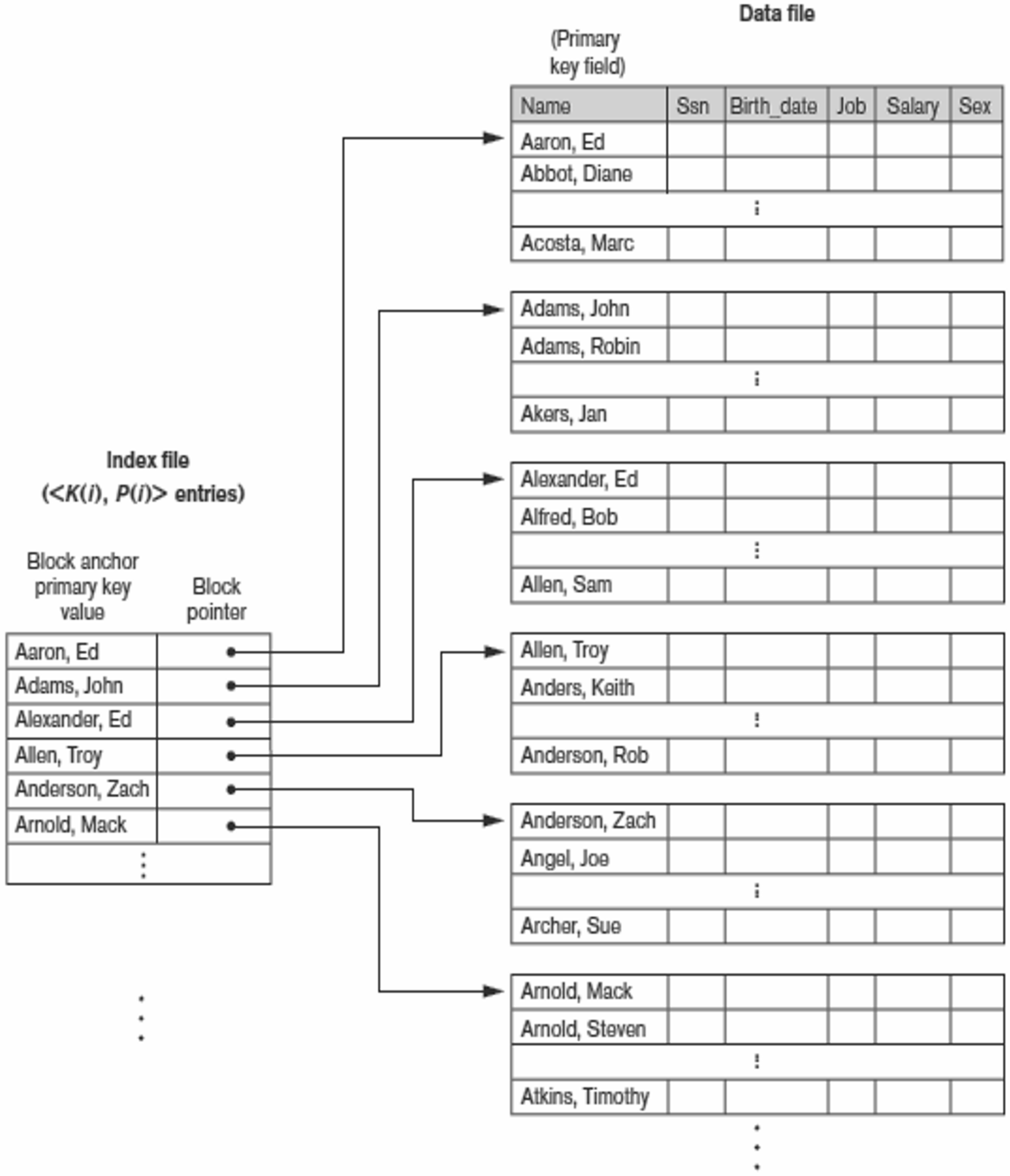
Type of Indexes

- Primary Indexes
- Clustering Indexes
- Secondary Indexes
- Multilevel Indexes
- Dynamic Multilevel Indexes Using B^+ Trees
- Indexes on Multiple Keys

- **Dense and Sparse**

Primary Index

- Is defined on an **ordered data file**.
  - The data file is ordered on a key field.
  - The key field is generally the primary key of the relation.
  - One index entry in the index file for each block in the data file
  - Indexes may be **dense or sparse**
    - Dense index has an index entry **for every search key value in the data file**
    - **Sparse index** has entries for only **some** search values



Primary Index

Data Access – Files vs. Primary Index

Example:

- **Consider an ordered file with r = 30,000 records stored on a disk with blocks of B = 1024 bytes. Records have a fixed size of R = 100 bytes .**
  - The **blocking factor** is bfr = **(B/R)** = (1024/100)= **10 records per block**.
  - The **number of blocks required** for the file is **b = (r/bfr)** = (30,000/10) = **3000 blocks.**
  - **Binary search** over the file requires a maximum of (**$\log_2 b$**)= ($\log_2 3000$) = **12 block accesses.**

Primary Index

Data Access – Files vs. Primary Index

- **Now consider constructing a primary index with an indexing attribute of V = 9 bytes and a block pointer of P = 6 bytes.**
  - **Each entry in the index file** occupies Ri = V + P = **15 bytes**.
  - **The number of entries in the index file** is ri = b = **3000 entries** (number of blocks in the data fx.)
  - **The blocking factor in the index file** is bfri = (B/Ri) = (1024/15)= **68 records per block**.
  - The number of blocks required for the index file is bi = (ri/bfri) **= 45 blocks**.
  - **Binary search over the index file** requires a maximum of ($\log_2$bi) = ($\log_2 45$) = **6 block accesses.**

- **To find a record using the index file, a maximum of 6 + 1 = 7 accesses to disk blocks is required, that is**, 5 hits less **than without the index file.**

Dense and Sparse Index

- **Dense index** — Index record appears for every search-key value in the file.
  - E.g. index on *ID* attribute of *instructor* relation

10101	
12121	
15151	
22222	
32343	
33456	
45565	
58583	
76543	
76766	
83821	
98345	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sparse Index

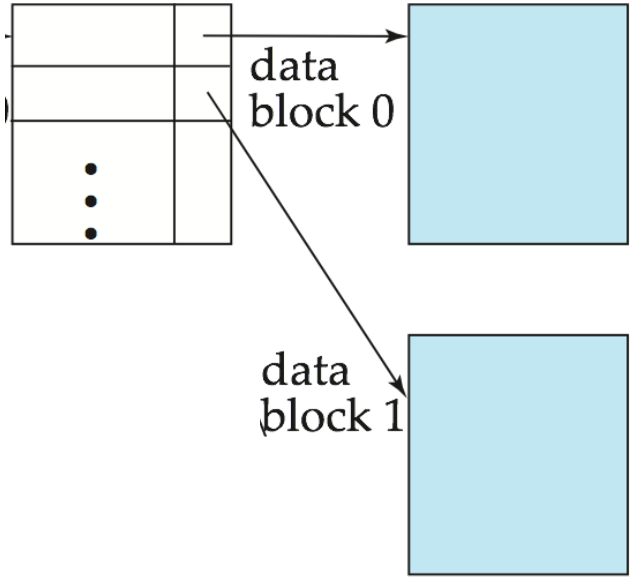
- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points

10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sparse Index

- Compared to dense indexes:
  - **Less space** and **less maintenance** overhead for **insertions and deletions**.
  - Generally **slower than dense index for locating records.**

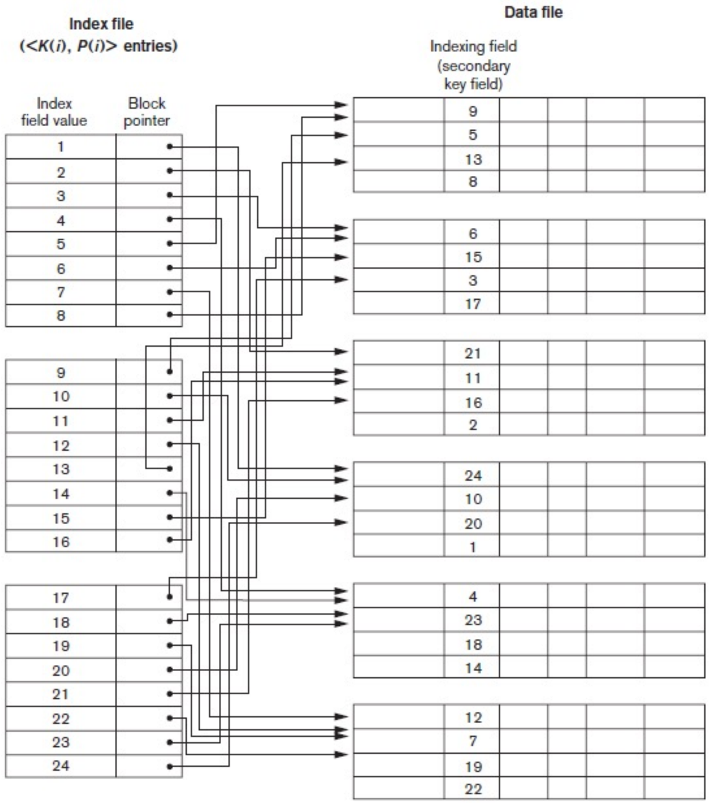


Secondary Index

- Secondary index may be generated from a field which is a candidate key and has **a unique value** in every record, or a **non-key with duplicate values.**

- A secondary index does **not determine the organization of the data file.**

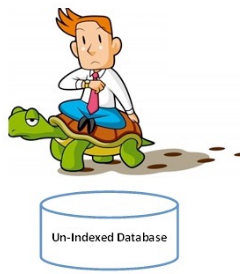
- Usually need more storage space and longer search time than primary index
  - Improved search time for arbitrary record



Dense secondary index (with block pointers) on a nonordering key field of a file.

Secondary Index

Data Access – Files vs. Secondary Index



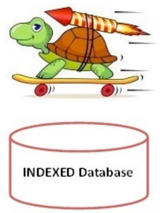
- Consider a file with r = 30,000 records stored on a disk with blocks of B = 1024 bytes. Records have a fixed size of R = 100 bytes .
- The **blocking factor is** bfr = (B/R) = (1024/100) = **10 records per block**.
  - **The number of blocks required** for the file is b = (r/bfr) = (30,000/10) **= 3000 blocks.**
- <u>Linear search</u> over the file requires a maximum of (b/2) = (3000/2) = **1500 block accesses.**

Secondary Index

- Consider constructing a secondary index with an indexing attribute of V = 9 bytes and a pointer to blocks of P = 6 bytes.

    - **Each entry in the index** file occupies Ri = V + P = 15 bytes.

Porquê?



    - **The number of entries** in the index file is ri = b **= 30000 entries.**
    - **The blocking factor** in the index file is bfri = (B/Ri) 0 (1024/15) = **68 records per block.**
    - **The number of blocks required** for the index file is bi = (ri/bfri) **= 442 blocks.**
    - **Binary search over the index file** requires a maximum of ($\log_2$bi) = ($\log_2$442) = **9 block accesses**.

- **To find a record using the index fi**le it takes a maximum of **9 + 1 = 10 accesses,** 1490 fewer accesses than without the index file

Secondary Index

How to choose secondary indexes

Add a secondary index **to a foreign key if it is frequently accessed.**

Add a secondary index **to any attribute that is frequently used as a search key.**

Add a secondary index **to any attributes frequently used in order by, group by, min, max, avg**

E.g. age in Student if frequently want list in age order.

When not to use secondary indexes

If the relation **is small – not many rows**

If **the relation or attribute is frequently updated**

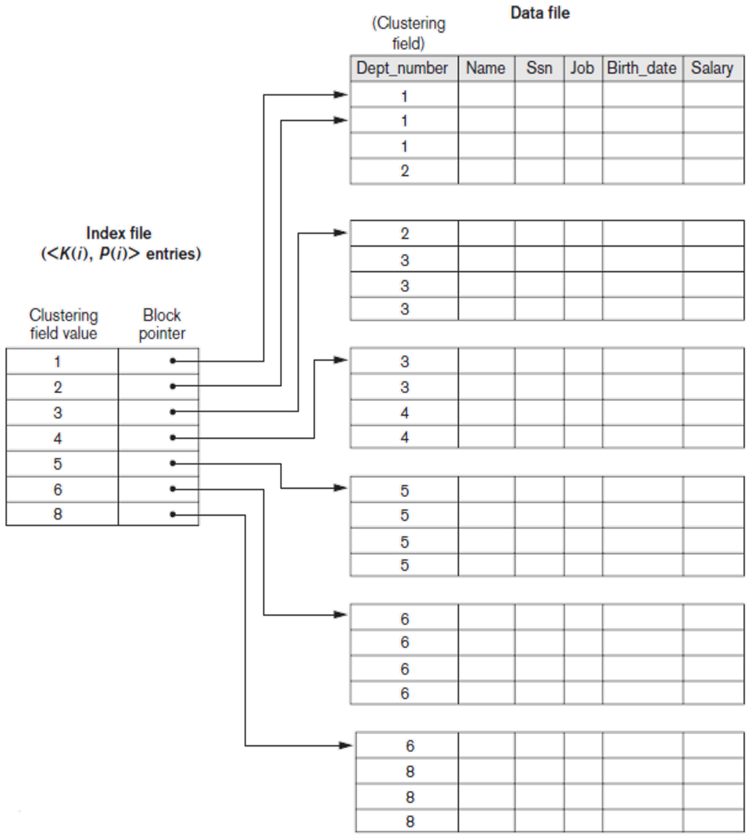
If periodic large updates, drop the index, update the data, re-create the index

If the attribute is always used in queries that retrieve a significant proportion of the rows in the relation

- E.g. If the attribute has low sparsity (i.e. the number of different values is small, such as gender F or M or …)

Clustering Index

- Clustering field
  - File records **are physically ordered on a nonkey field without** a distinct value for each record
- **Total Entries** in the index is **equal to the number of distinct values of the grouping attribute.**
- The biggest disadvantage of a cluster index is the insertion and delete of records because it **requires the ordering of the data file and the index file**



A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file

Properties of Index

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Table 17.1 Types of indexes based on the properties of the indexing field

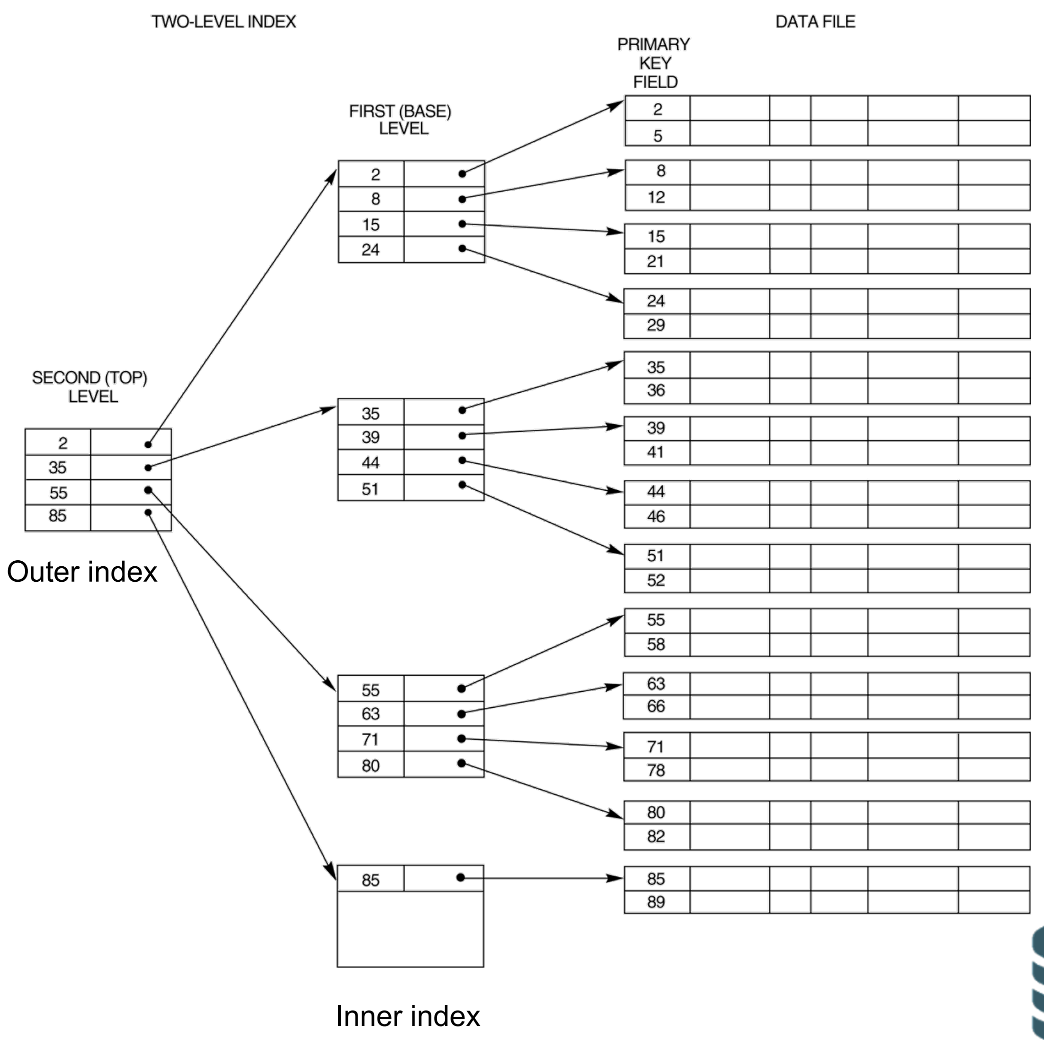
Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

Table 17.2 Properties of index types

Multilevel Indexes

- Because a single-level index is an ordered file, we can **create a primary index** *to the index itself* ; in this case, the **original index file is called the *first-level index*** and **the index to the index is called the *second-level index.***
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top-level* fit in one disk block
- A multi-level index can be created **for any type of first-level i**ndex (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block



Multilevel Indexes

- If primary index does not fit in memory, then treat primary index kept on disk as a sequential file and construct a sparse index on it.
  - outer index – a sparse index of primary index
  - inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- **Indices at all levels must be updated on insertion or deletion from the file.**

Multilevel Indexes

Data Access – Files vs. Multilevel Index

- Consider again the file with r = 30,000 records stored on a disk with blocks of B = 1024 bytes. Records have a fixed size of R = 100 bytes and do not traverse blocks.
- The **blocking factor** is bfr = (B/R)= (1024/100) = **10** records per block
- The **number of blocks required** for the file is b = (r/bfr) = (30,000/10) = **3000** blocks.
- **Binary search** on the file requires a maximum of ($\log_2$) = ($\log_2 3000$) = **12** block accesses.
- Consider again the **previous secondary index** with **b_1= 442 bl**ocks required for the index file and with blocking factor of **bfri = 68** records per block

Multilevel Indexes

Data Access – Files vs. Multilevel Index

- **Consider now a multilevel index from the secondary index.**
- Number of levels is ($\log_{bfri}$ r)) = ($\log_{68}$ (30000)) = 3 levels
- Levels
  - The number of blocks required **for the second level** is b2 = b1/bfri = 442/68 = 7 blocks.
  - The number of blocks required for the third level is b3 = b2/bfri = 7/68 = 1 block, i.e. the third level is the top level.

To find a record using the multiple-level index, it is necessary to access one block for each level plus the block of the records file, i.e., 3 + 1 = 4 accesses to disk blocks, which is at most 6 fewer accesses than with only the secondary index(10 accesses).