



DATABASE SYSTEMS

CS – 355/CE – 373

Instructor: Maria N. Samad

November 25th, 2024

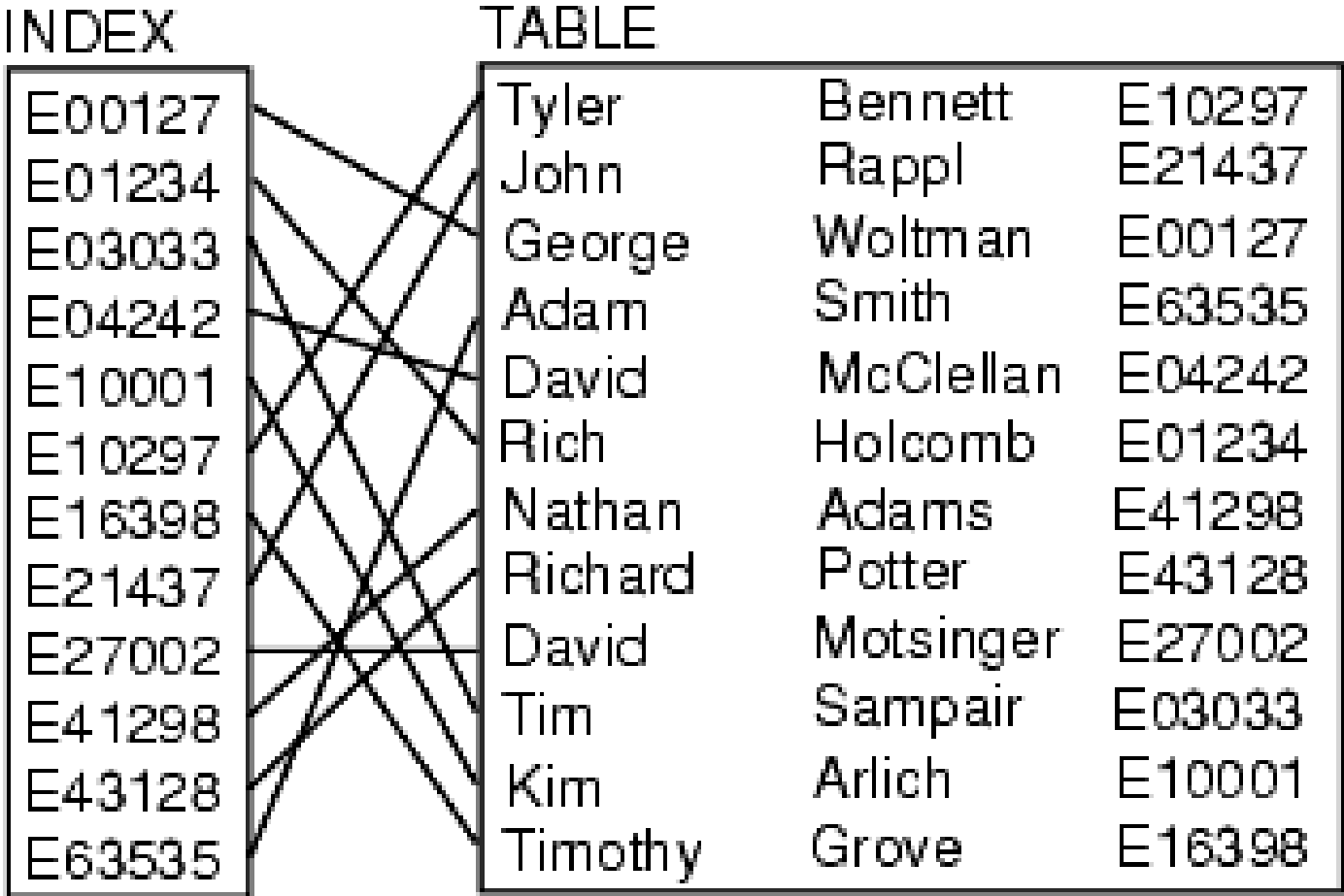
MOTIVATION

- Many queries reference only a small proportion of the records in a file.
- For example, a query such as
 - *“Find all instructors in the Physics department”, or*
 - *“Find the total number of credits earned by the student with ID 22201”*references only a fraction of the instructor or student records.
- It is **inefficient** for the system to read every tuple in the instructor relation to check if the dept name is *“Physics”*.
- To allow these forms of access **efficiently**, we design additional structures that we associate with files using **indexing**.

INDEXING

- An ***index*** for a file in a database system is a reference to the data with which we can jump to it directly, or to the area directly where that data might be present.
- It works in much the same way as the index in a textbook.
- Database-system indices use a search key to point to a specific location in order to retrieve a record.
- For example, to retrieve a student record given an ID, the database system would look up an index to find on which disk block the corresponding record resides.
- The disk block is then fetched, to get the appropriate student record.

A TYPICAL INDEX FILE



TYPES OF INDICES

- There are two basic kinds of indices:
 - ***Ordered indices:***
 - Based on a sorted ordering of the key values.
 - ***Hash indices:***
 - Based on a uniform distribution of values across a range of buckets.
 - The bucket to which a value is assigned is determined by a function, called a hash function.
- There are multiple techniques for both kinds of indices, and no one technique is defined as the best.
- Instead, each technique is evaluated on the basis of some relevant factors.

PARAMETERS TO CONSIDER IN INDEXING

- A given indexing technique must be evaluated against the following parameters:
 - ***Access types:***
 - The types of data access that are supported efficiently
 - This may include accessing records with a specific search attribute, or finding records with a range of values, etc
 - ***Access time:***
 - The time it takes to find a particular data item, or set of items
 - Depending on how fast and efficient the desired search is, the most appropriate indexing technique may be used

PARAMETERS TO CONSIDER IN INDEXING

- A given indexing technique must be evaluated against the following parameters:
 - ***Insertion time:***
 - The time it takes to insert a new data item
 - This includes looking for the correct place in the data set to insert the value, the time to do the insertion, as well as time to update the index data structure
 - ***Deletion time:***
 - The time it takes to delete a data item
 - This includes both the time to find the item and removing it, as well as the time to update the index data structure

PARAMETERS TO CONSIDER IN INDEXING

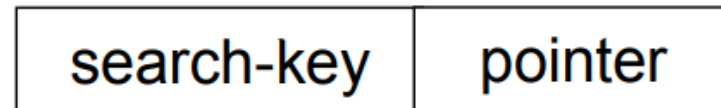
- A given indexing technique must be evaluated against the following parameters:
 - ***Space Overhead:***
 - The additional space occupied by an index structure will also need to be considered when choosing the indexing technique
 - However, it is the least important choice for many, because in many cases, we can get faster results at the cost of extra space

SEARCH KEY

- An attribute or set of attributes that are used to reference records in a file
- Nothing to do with primary key, candidate key or superkey
- ***Search key*** may or may not be a primary key
- Once a search key is selected, it doesn't mean we cannot have any other attributes as search keys in the same relation.
- This varies from program to program

INDEX STRUCTURE

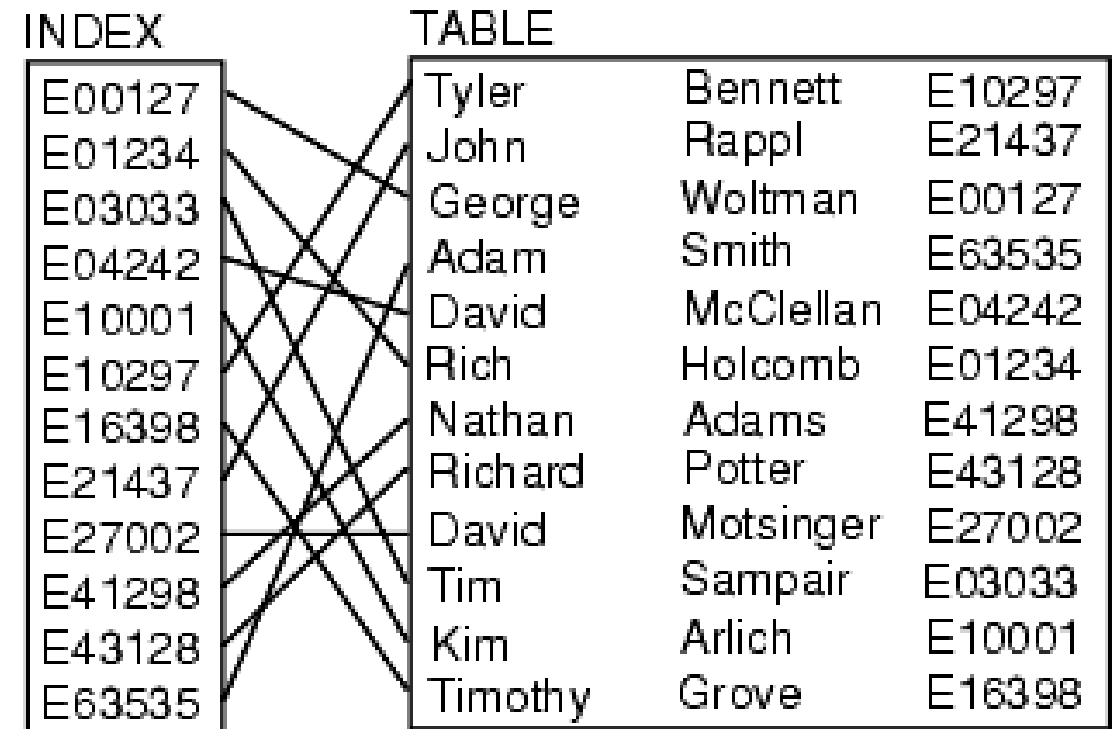
- An index structure typically consists of:
 - A search key
 - An associated pointer to the disk block (or secondary storage) corresponding to the required tuple.



- Index files are typically much smaller than source files.

ORDERED INDEX

- An **ordered index** stores the values of the search keys in sorted order.
- It associates with each search key the records that contain the key.
- The records in the indexed file may themselves be stored in some sorted order, (not necessarily the order of the search keys in the index structure).
- A file may have several indices, each on a different search key.



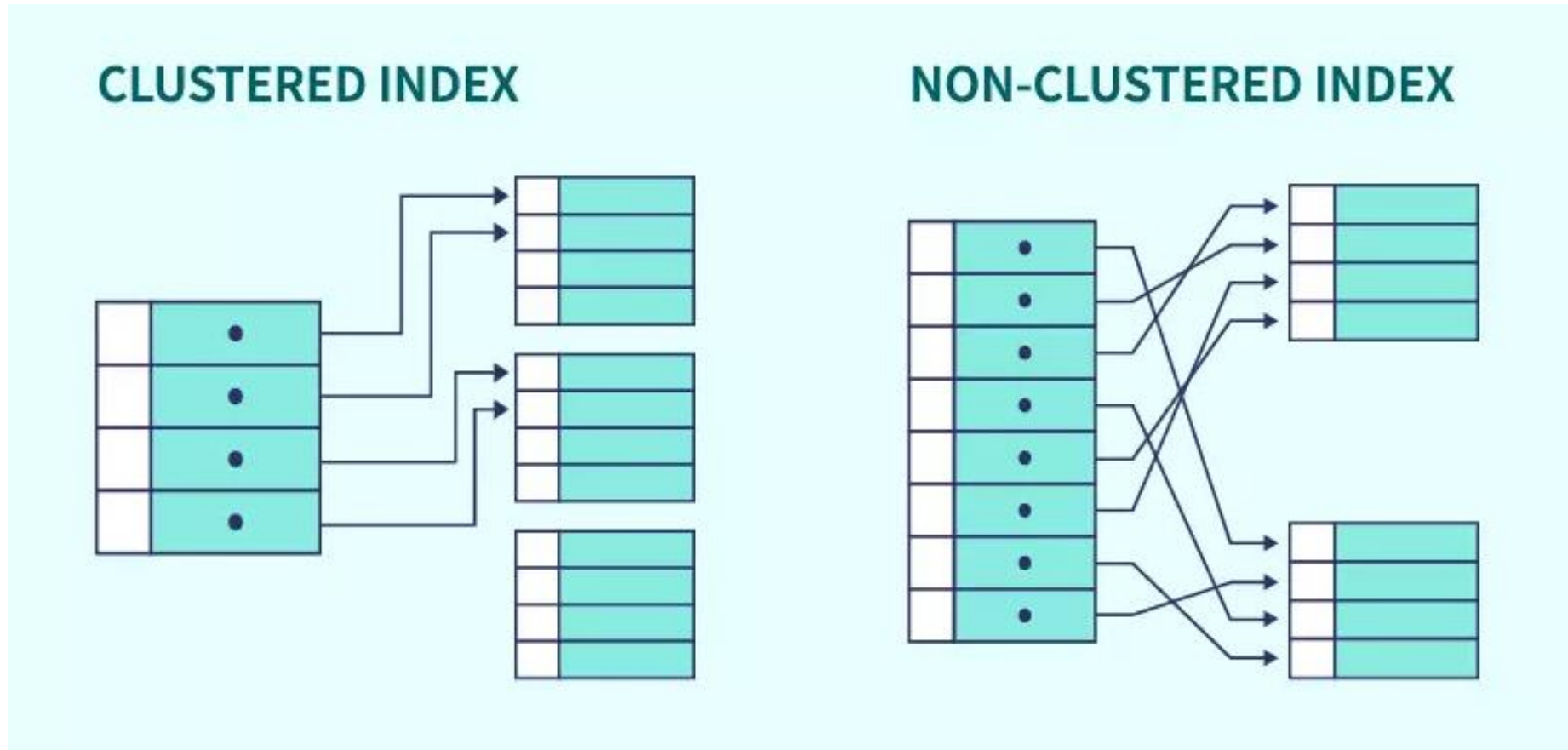
ORDERED INDEX – TERMINOLOGY

- In an ordered index, index entries are stored in a sorted manner on the search key value.
- ***Clustering index:***
 - In a sequentially ordered file, it is the index whose search key specifies the sequential order of the file.
 - This is also called the ***primary index***.
 - The search key of a primary index is usually (but not necessarily) the primary key.
 - For example, in the instructor table, a clustered index can be on search key of instructor ID (i.e. primary key), or on salary, which is not the primary key
 - Both however are considered clustering indices because records can be grouped together i.e. ***clustered*** and sequentially sorted based on their values

ORDERED INDEX – TERMINOLOGY

- ***Non-Clustering index:***
 - An index whose search key specifies an order different from the sequential order of the file.
 - Also called ***secondary index***.
 - For example, in the instructor table, a non-clustered index can be on search key of department, or instructor name, which is not sequential but can still be ordered, hence non-clustering
 - Examples:
 - Destination or Flight number in airline reservation system
 - Email or username in a Social Media Platform
 - Title or author name in library management system
 - All these examples can be ordered, which are not sequential, hence non-clustering or secondary index

CLUSTERED AND NON-CLUSTERED INDICES



TYPES OF ORDERED INDICES

- There are two types of ordered indices:
 - *Dense Index*
 - *Sparse Index*
- No matter which index type is used, they will always be defined in the format, as discussed before, i.e.

search-key	pointer
------------	---------

DENSE INDEX

- In a ***dense index***, an index entry appears for every search-key value in the file.
- This has a one-to-one direct relation between the index and the record in the table
- Each index will include a pointer or reference to the first data record with that search key
- This means there will be as many entries in the index file, as there are values of that particular search key.
- For example, if the search key is instructor ID, and there are, let's say 10 IDs in the instructor table, then the index file will have 10 entries – one for each record
- Similarly, if the search key is instructor name, and there are, let's say 5 unique names in the instructor table, then the index file will have 5 entries

DENSE INDEX

- In a ***dense clustering index***, the index record contains the search-key value and a pointer to the first data record with that search-key value.
- The rest of the records with the same search-key value would be stored sequentially after the first record, since, because the index is a clustering one, records are sorted on the same search key.
- If the search-key value is a unique value like primary key, then it will be a complete one-to-one relation between index file and database relation
 - The index file will have the same number of entries, as there are records in the table
- In a ***dense non clustering index***, the index must store a list of pointers to all records with the same search-key value, and not just to the first record with that value.

DENSE INDEX – EXAMPLE

Search Activity: Find the instructor record with the ID 22222

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

Figure 14.2 Dense index.

DENSE INDEX – EXAMPLE

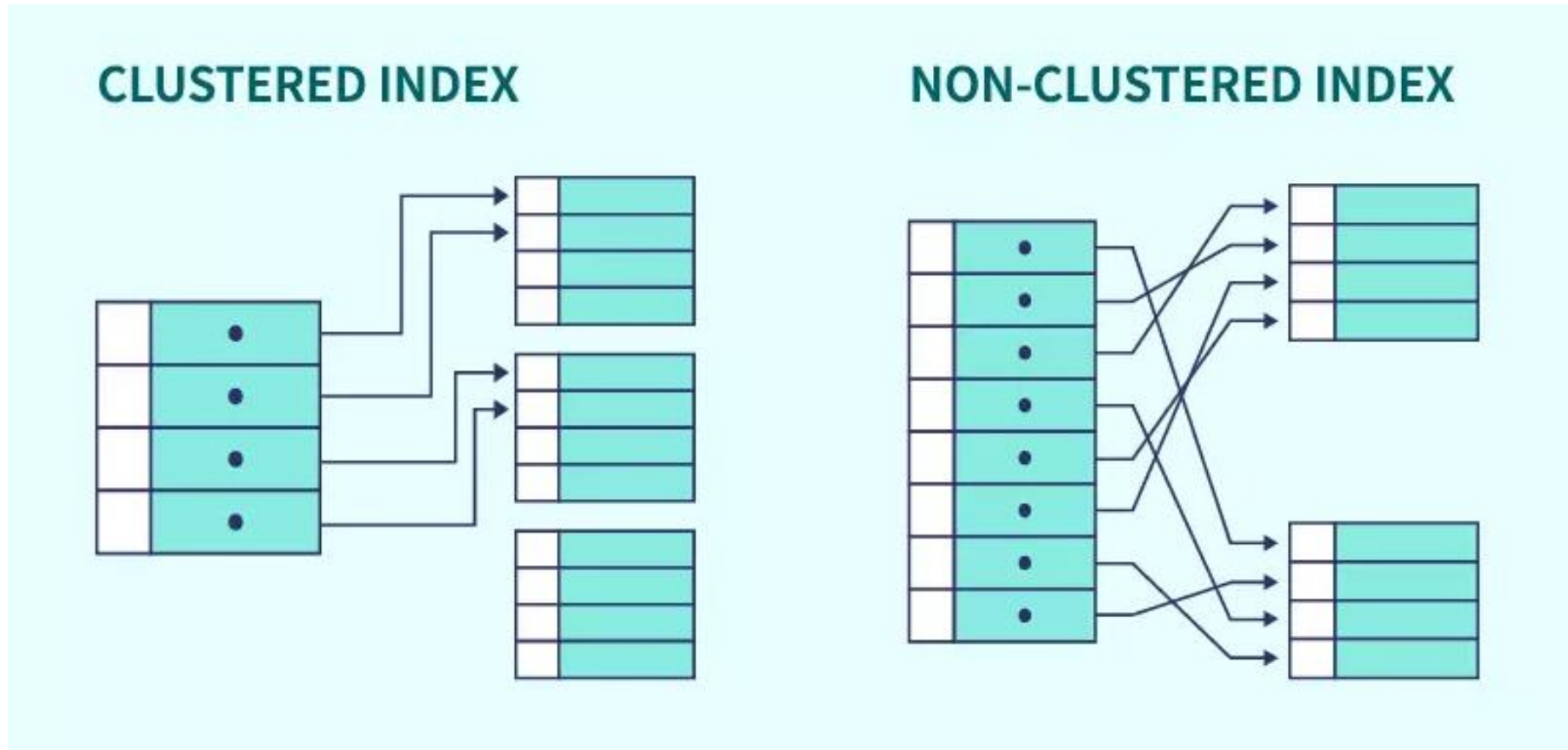
- ***Dense Clustering Index:***

- Using salary as the search key
- The same salaries must be grouped together in the table
- The index record will contain the search key value, i.e. salary values, and the pointer to the first record of that salary value

- ***Dense Non-clustering Index:***

- Using salary as the search key
- The salaries are not grouped together in the table
- The index record will contain the search key value, i.e. salary value, with multiple pointers to each of the records containing salary values

CLUSTERED AND NON-CLUSTERED INDICES



SPARSE INDEX

- In a ***sparse index***, an index entry appears for only some of the search-key values.
- Sparse indices can be used **only** if the relation is stored in sorted order of the search key; that is, **if the index is a clustering index**.
- Each index entry contains a search-key value and a pointer to the first data record with that search-key value.
- To locate a record, we find the index entry with the largest search-key value that is less than or equal to the search-key value for which we are looking.
- We start at the record pointed to by that index entry and follow the pointers in the file until we find the desired record.

SPARSE INDEX – EXAMPLE

Search Activity: Find the instructor record with the ID 22222

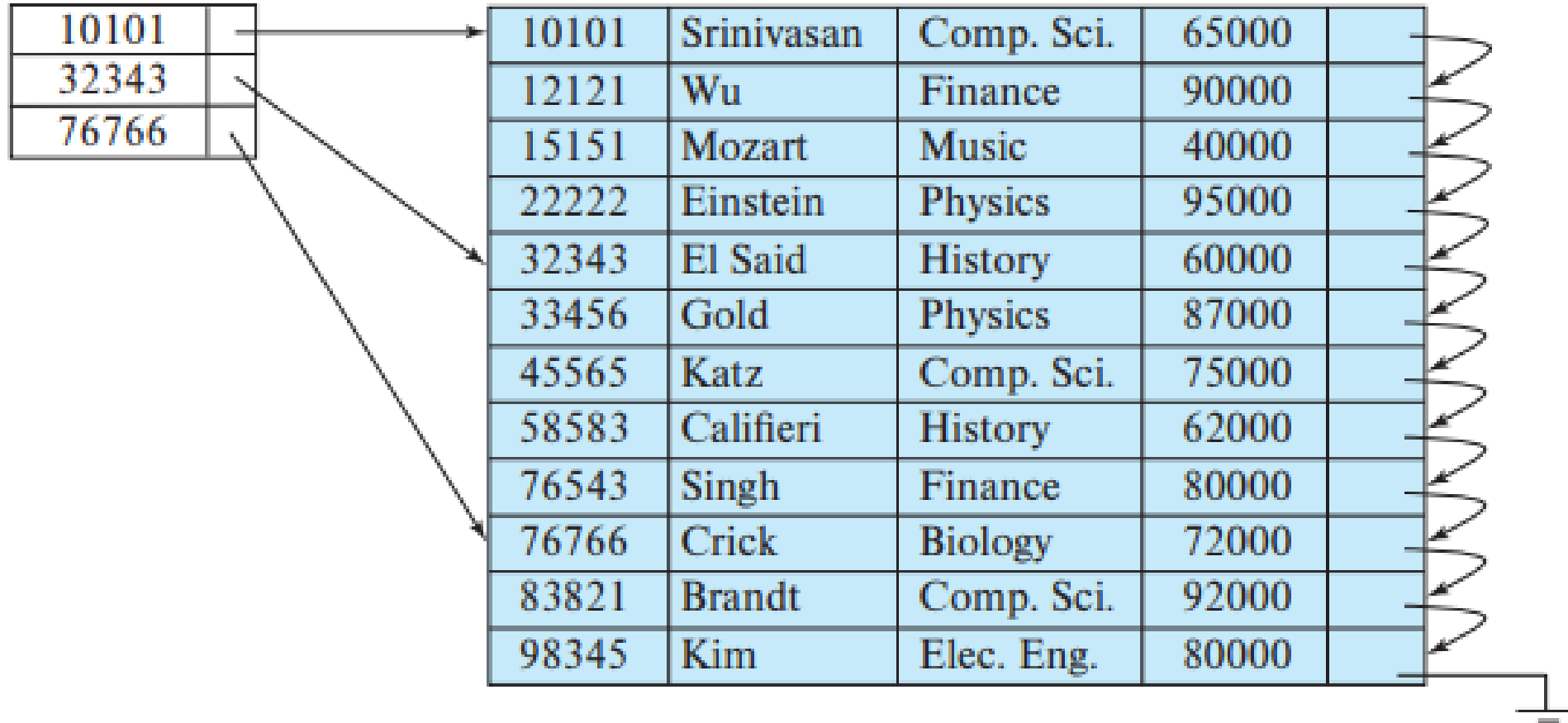
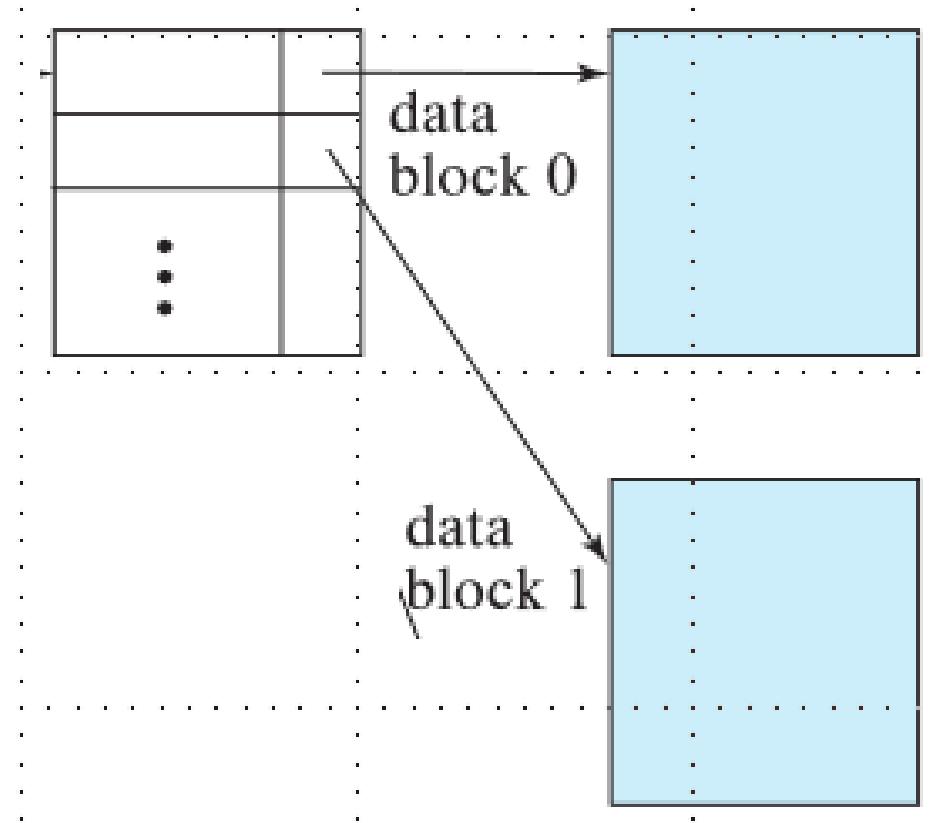


Figure 14.3 Sparse index.

DENSE VS SPARSE INDICES

- It is generally faster to locate a record if we have a dense index, rather than a sparse index.
- However, sparse indices have advantages over dense indices in that they require less space and they impose less maintenance overhead for insertions and deletions.
- A good compromise is to have a sparse index with one index entry per block.
- The reason this design is a good trade-off is that the dominant cost in processing a database request is the time that it takes to bring a block from disk into main memory.
- Once we have brought in the block, the time to scan the entire block is negligible.

SPARSE INDEX WITH ONE INDEX ENTRY PER BLOCK



ORDERED INDICES EXAMPLE

- Example: Given a table for books in library management system with the following sample:

BookID	Title	Author	Publication Year	Genre
101	The Great Gatsby	F. Scott Fitzgerald	1925	Fiction
102	This Side of Paradise	F. Scott Fitzgerald	1920	Romance
103	The curious case of Benjamin Button	F. Scott Fitzgerald	1922	Mystery
104	To Kill a Mockingbird	Harper Lee	1960	Thriller
105	A Brief History of Time	Stephen Hawking	1988	Science
106	Theory of Everything	Stephen Hawking	2002	Documentary
107	Inferno	Dan Brown	2013	Mystery
108	The Da Vinci Code	Dan Brown	2003	Mystery
109	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy
110	The Running Grave	Robert Galbraith	2023	Thriller

ORDERED INDICES EXAMPLE

- Answer the following:
 1. What is the primary/clustered index for the given table?
 2. What can be the possible secondary/non-clustered indices in the given table?
 3. Design an index structure for a dense clustered index
 4. Design an index structure for a dense non-clustered index
 5. Design an index structure for a sparse index
 6. For the sparse index designed in part 5, write the steps in order to find the one of the entries not directly accessible by the index in the sparse index structure

ORDERED INDICES EXAMPLE

- Answer the following:
 1. What is the primary/clustered index for the given table?
 - Both BookID and Author name are grouped together
 - However, author name is not ordered at all
 - Therefore, the only clustered index will be BookID for this particular form of the Book table

ORDERED INDICES EXAMPLE

- Answer the following:
 2. What can be the possible secondary/non-clustered indices in the given table?
 - Title, Publication Year and Genre values are not “grouped together” so they might be considered as non-clustered
 - But confirm if there can be some order in them? Yes, then any one can be chosen for the secondary index
 - If we make Author name as a non-clustering index, that will lead to discrepancies, because non-clustering index requires a separate pointer to all values, which means we have separate pointers for the same value being repeated, and this will cause the search to not work properly

ORDERED INDICES EXAMPLE

- Answer the following:

3. Design an index structure for a dense clustered index

Search Key	Pointer	BookID	Title	Author	Publication Year	Genre
101	—	▶ 101	The Great Gatsby	F. Scott Fitzgerald	1925	Fiction
102	—	▶ 102	This Side of Paradise	F. Scott Fitzgerald	1920	Romance
103	—	▶ 103	The curious case of Benjamin Button	F. Scott Fitzgerald	1922	Mystery
104	—	▶ 104	To Kill a Mockingbird	Harper Lee	1960	Thriller
105	—	▶ 105	A Brief History of Time	Stephen Hawking	1988	Science
106	—	▶ 106	Theory of Everything	Stephen Hawking	2002	Documentary
107	—	▶ 107	Inferno	Dan Brown	2013	Mystery
108	—	▶ 108	The Da Vinci Code	Dan Brown	2003	Mystery
109	—	▶ 109	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy
110	—	▶ 110	The Running Grave	Robert Galbraith	2023	Thriller

Dense Clustered Index File

ORDERED INDICES EXAMPLE

- Answer the following:
 4. Design an index structure for a dense non-clustered index
 - Assume search key is Genre
 - What is wrong in the following structure?
 - Not an ordered index file

Search Key	Pointer	BookID	Title	Author	Publication Year	Genre
Fiction	—	▶ 101	The Great Gatsby	F. Scott Fitzgerald	1925	Fiction
Romance	—	▶ 102	This Side of Paradise	F. Scott Fitzgerald	1920	Romance
Mystery	—	▶ 103	The curious case of Benjamin Button	F. Scott Fitzgerald	1922	Mystery
Thriller	—	▶ 104	To Kill a Mockingbird	Harper Lee	1960	Thriller
Science	—	▶ 105	A Brief History of Time	Stephen Hawking	1988	Science
Documentary	—	▶ 106	Theory of Everything	Stephen Hawking	2002	Documentary
Fantasy	—	▶ 107	Inferno	Dan Brown	2013	Mystery
		▶ 108	The Da Vinci Code	Dan Brown	2003	Mystery
		▶ 109	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy
		▶ 110	The Running Grave	Robert Galbraith	2023	Thriller

Dense Non-Clustered Index File

ORDERED INDICES EXAMPLE




- Answer the following:
 4. Design an index structure for a dense non-clustered index
 - Assume search key is Genre
 - Assuming alphabetical order for Genre, the correct index structure will be like:
 - I have color-coded it to understand better, otherwise no need for colors as such

Search Key	Pointer	BookID	Title	Author	Publication Year	Genre
Documentary		101	The Great Gatsby	F. Scott Fitzgerald	1925	Fiction
Fantasy		102	This Side of Paradise	F. Scott Fitzgerald	1920	Romance
Fiction		103	The curious case of Benjamin Button	F. Scott Fitzgerald	1922	Mystery
Mystery		104	To Kill a Mockingbird	Harper Lee	1960	Thriller
Romance		105	A Brief History of Time	Stephen Hawking	1988	Science
Science		106	Theory of Everything	Stephen Hawking	2002	Documentary
Thriller		107	Inferno	Dan Brown	2013	Mystery
		108	The Da Vinci Code	Dan Brown	2003	Mystery
		109	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy
		110	The Running Grave	Robert Galbraith	2023	Thriller

Dense Non-Clustered Index File

ORDERED INDICES EXAMPLE

- Answer the following:
 5. Design an index structure for sparse index
 - Can only be applied to clustering indices
 - As explained in Question 1, only BookID in the given sample can be treated as a clustering index, so generating one tentative index structure for Sparse Index on BookID

Search Key	Pointer
101	
104	
107	

Sparse Index File

BookID	Title	Author	Publication Year	Genre
101	The Great Gatsby	F. Scott Fitzgerald	1925	Fiction
102	This Side of Paradise	F. Scott Fitzgerald	1920	Romance
103	The curious case of Benjamin Button	F. Scott Fitzgerald	1922	Mystery
104	To Kill a Mockingbird	Harper Lee	1960	Thriller
105	A Brief History of Time	Stephen Hawking	1988	Science
106	Theory of Everything	Stephen Hawking	2002	Documentary
107	Inferno	Dan Brown	2013	Mystery
108	The Da Vinci Code	Dan Brown	2003	Mystery
109	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy
110	The Running Grave	Robert Galbraith	2023	Thriller

ORDERED INDICES EXAMPLE

- Answer the following:
 6. For the sparse index designed in part 5, write the steps in order to find the one of the entries not directly accessible by the index in the sparse index structure
 - Assume, we are looking for a book ID = 109
 - The steps for search will be:
 - Search for search key = 109
 - It is not in the index file, so look for the largest value that is immediately smaller than 109. In this case it is 107. Go to the record pointed by this search key from the index file
 - Linearly search for entries:
 - It finds the result in the second row after 107
 - PLEASE NOTE: If there are multiple search criteria, we can look for it depending on the query being used. Don't forget the difference of "and" and "or" in conditions

ORDERED INDICES

- Class Activity

ORDERED INDICES

- Class Activity Solution:
 - [Ordered Indices Solution](#)