

INFS7901

Database Principles

Indexing

Fundamentals of Database Systems (Sixth Edition) Chapter 18

Hassan Khosravi & Ash Rahimi

Course Outline

Relational Databases (Most of INFS1200)

Duration	6 weeks
Main Topics	Relational Database Design, Relational Model, Functional Dependencies and Normal Forms, SQL

Data Access Methods (An intro to many of the topics covered in COMP3506)

Duration	4 weeks
Main Topics	Asymptotic Analysis, Searching and Sorting, Abstract Data Types (Dictionary), Binary Search Trees, Hashing

Indexing and Query Processing (Some of the topics covered in INFS2200)

Duration	2 weeks
Main Topics	Tree- and Hash-Based Indexes, Relational Algebra, Query Optimization,

Notes

- **Lecture:**

- Use <https://visualgo.net> to get a better sense of the algorithms from Module 2.
- Starting Module 3 today – Short module on Indexing and a longer module on query optimization.

Learning Objectives

Description	Tag
Explain the benefits of using indexes in database systems.	Indexing
Describe the notion of a dense index.	
Describe the notion of a sparse index.	
Describe the properties of primary indexes.	
Describe the properties of clustering indexes.	
Describe the properties of secondary indexes.	
Describe the properties of multi-level indexes.	
Describe the properties of tree-structured indexes.	
Explain how a tree-structured index supports both range and point queries.	
Describe the properties of hash-based indexes.	
Explain why hash-based index only support point queries.	
Compare and contrast the performance of different indexes.	
Given information about the most common queries in a database, recommend which attributes using which methods should be indexed.	

Intro to Indexing

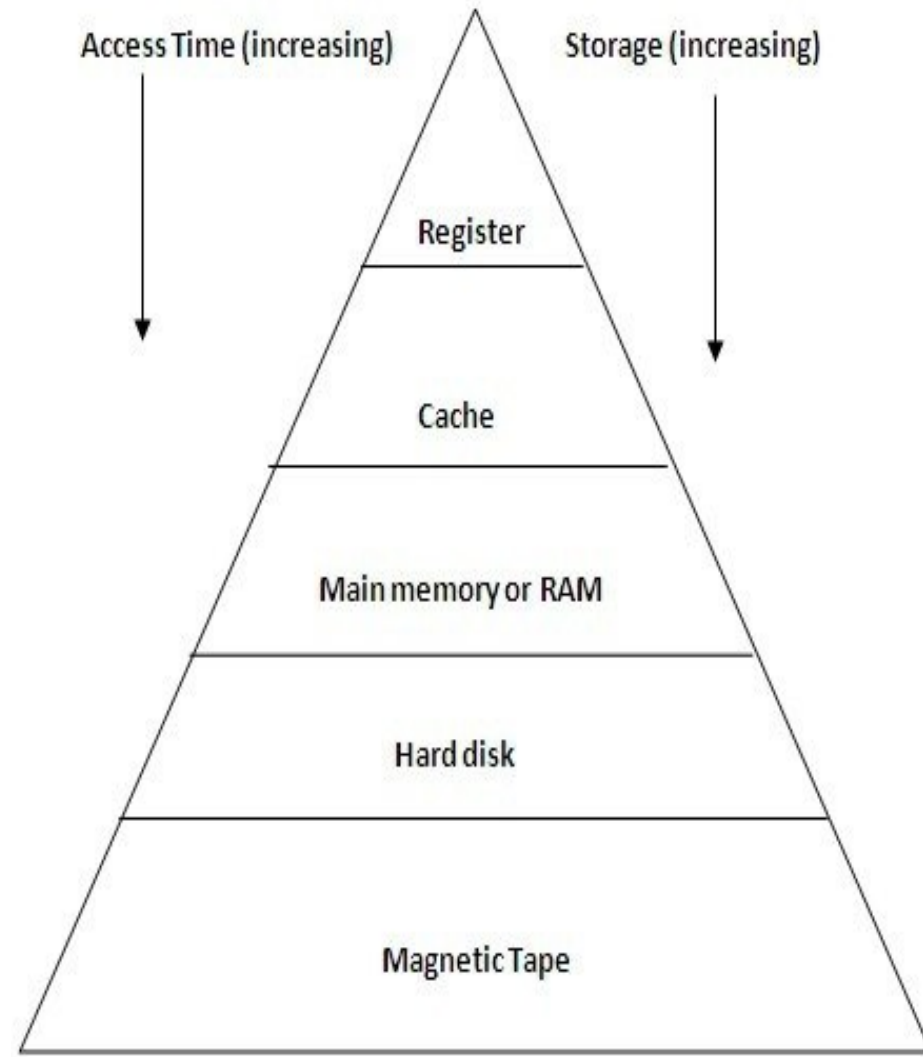
Single-level Indexes

Multi-level Indexes

Indexes on Multiple Keys

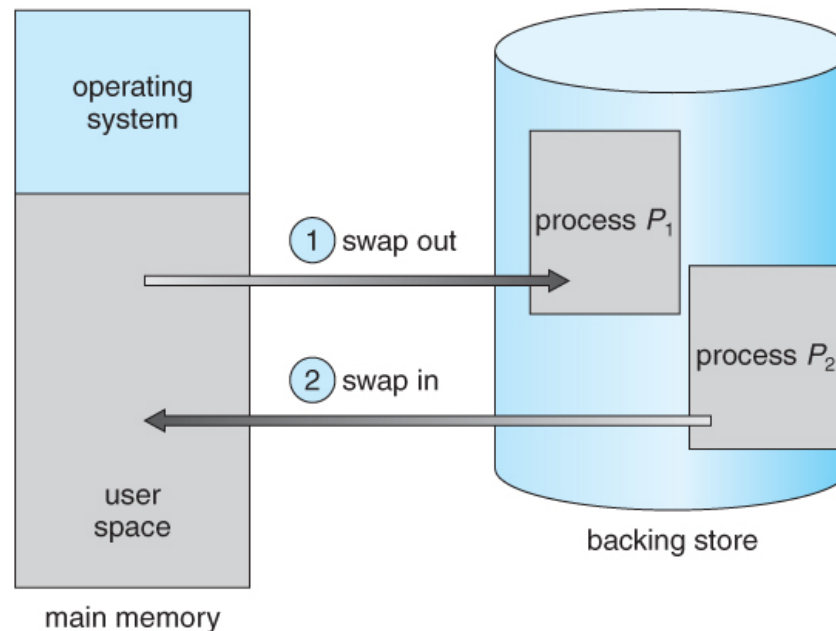
Storage Hierarchy

- **Internal register** is for holding the temporary results and variables.
- **Cache** is used by the CPU for memory which is being accessed repeatedly.
- **RAM** is used to stores data and machine code currently being used.
- **Hard disk** is used to keep data permanently in this memory - not directly accessed by the CPU.
- **Magnetic tape** is usually used for backing up large data.



Operating System - Memory Management

- **Memory management**, a form of **resource management**, is the process of controlling and coordinating computer memory to optimize overall system performance.



File Organization

- **Primary file organizations:** determine how the file records are physically placed on the disk
 - unordered file: adding records as they are inserted. Also called a heap file.
 - Ordered file: ordered by value of a specific field.
- **Secondary organizations:** allow efficient access to file records based on alternate fields than those that have been used for the primary file organization.

Ordered File vs. Unordered File

	<i>insert</i>	<i>Search order key</i>	<i>Search Non-order key</i>	<i>delete</i>
• Ordered File	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$
• Unordered File	$O(1)$	$O(n)$	$O(n)$	$O(n)$

Indexing

- Index is an **auxiliary** file that makes it more efficient to search for a record in the data file.
- The index is *usually* specified on one field of the file.
- The index is called an access path on the field.
 - A **dense index** has an index entry for every search key value (and hence every record) in the data file.
 - A **sparse (or nondense) index** has index entries for only some of the search values

Index Structures

- Types of Single-level Indexes

- Primary Indexes
- Clustering Indexes
- Secondary Indexes

	Ordered File	Unordered File
Key	Primary Index	Secondary Index
Non-key	Clustering Index	Secondary Index

- Multilevel Indexes

- Index on index

- Indexes on Multiple Keys

Intro to Indexing

Single-level Indexes

Multi-level Indexes

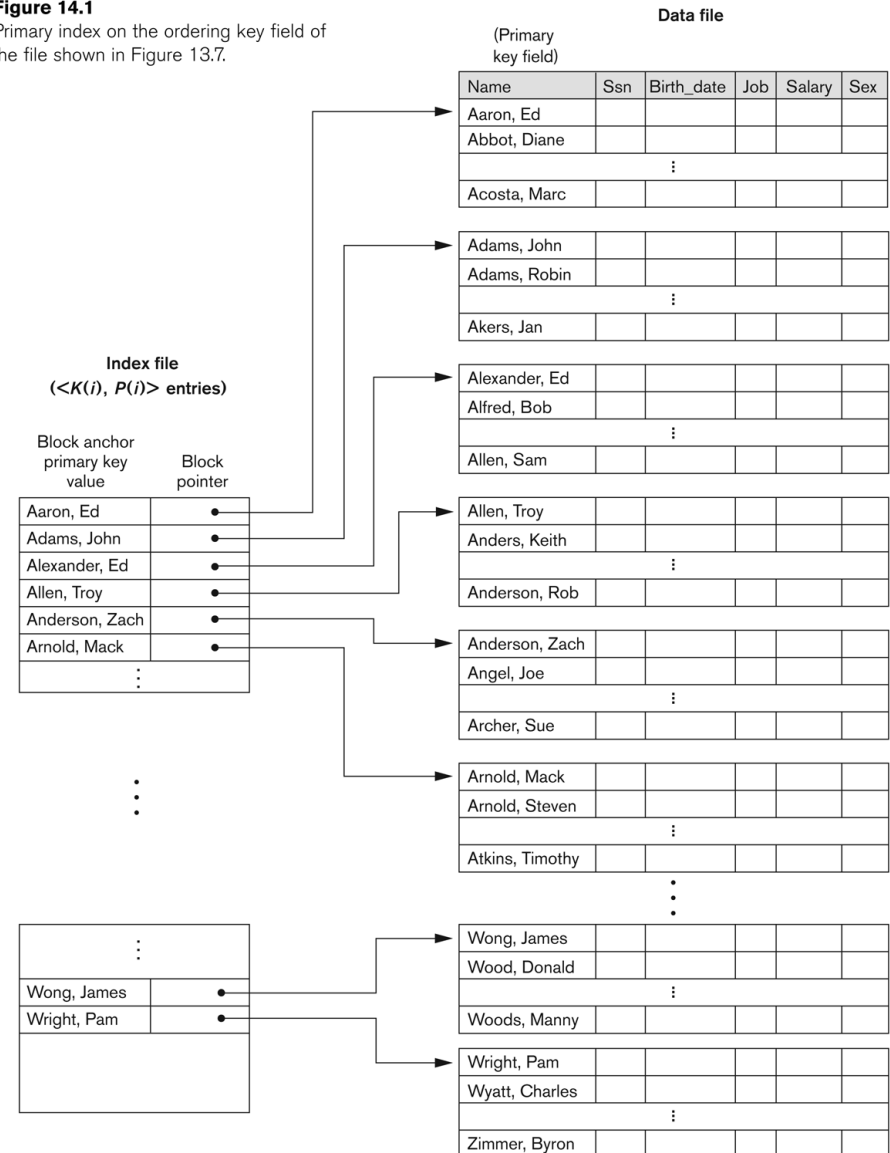
Indexes on Multiple Keys

Primary Index

- Defined on an **ordered** data file, which is sorted using a **key** field.
- A **primary index** is an ordered file with 2-field entries $\langle K(i), P(i) \rangle$
 - $K(i)$: is the key field
 - $P(i)$: is a pointer to the record
- A primary index is a **sparse** index

Figure 14.1

Primary index on the ordering key field of the file shown in Figure 13.7.



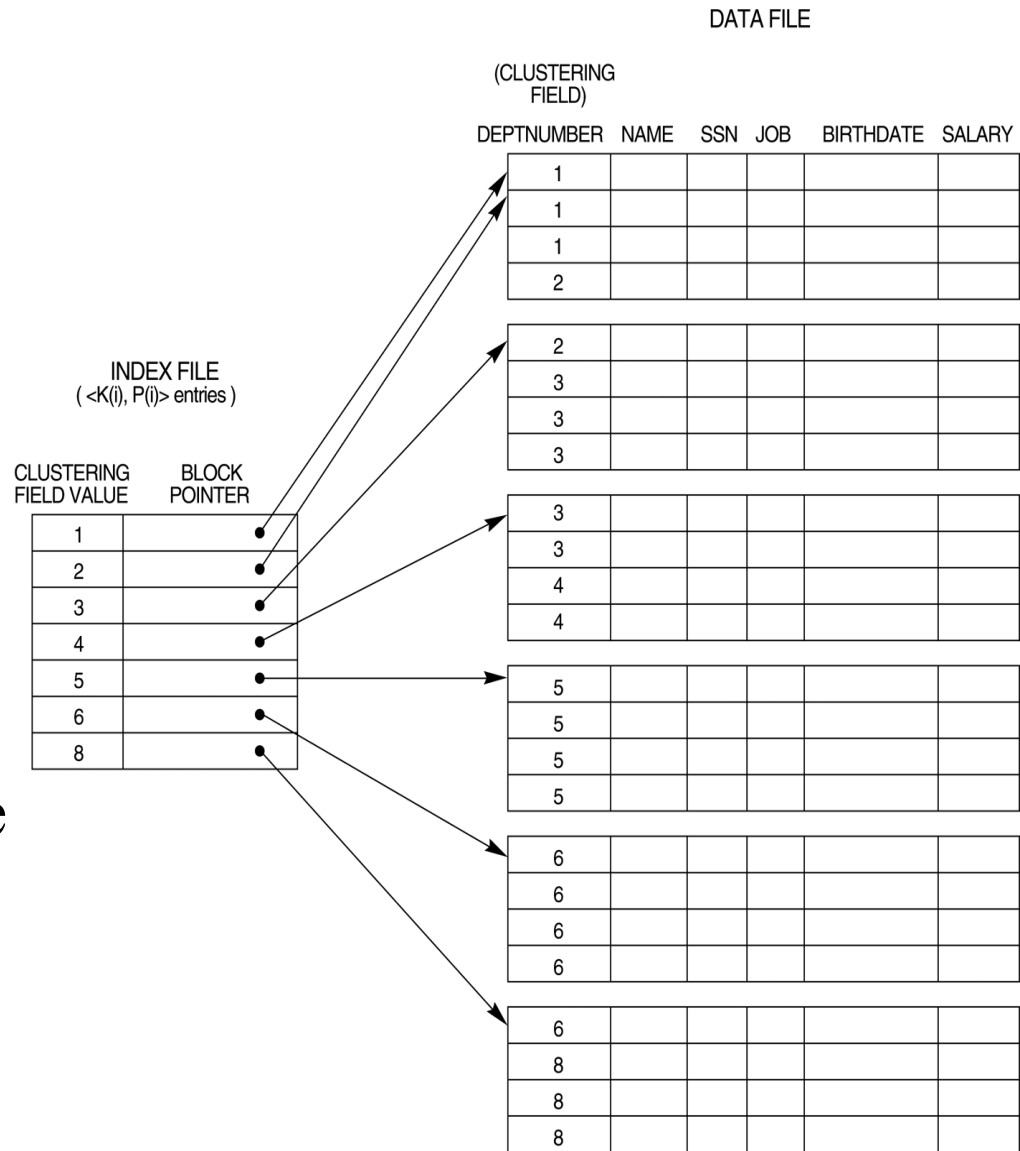
Primary Index vs. Data File

- A primary index occupies much less space than a data file because:
 - It is sparse, so few index entries
 - Each entry has only two columns

Even though both will be using Binary Search, searching a primary index is much faster than search the data file

Clustering Index

- Defined on an **ordered** data file, which is sorted using a **non-key** field.
- A **clustering index** is an ordered file with 2-field entries $\langle K(i), P(i) \rangle$
 - $K(i)$: is the ordered field
 - $P(i)$: is a pointer to the first block with that value
- A clustering index is a **sparse** index



Primary Indexing Question

- “There is no point in having a primary index on top of an ordered data file, which is sorted using a key field. We can already use binary search to find things in $O(\lg n)$ ”
- Do you agree or disagree with the statement above. Justify your answer

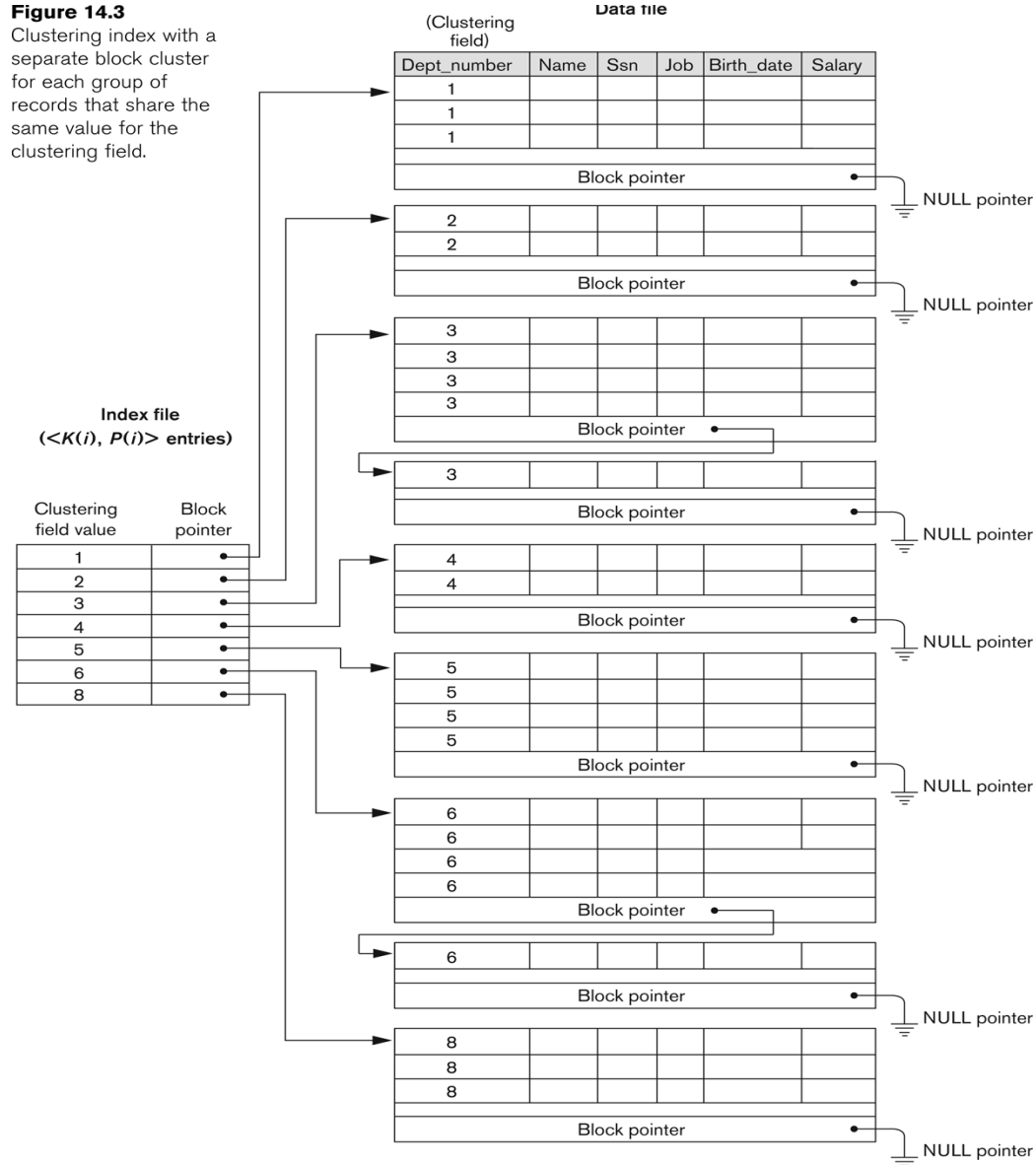
Disagree because:

- A primary index occupies much less space than a data file because it is sparse (therefore size is no longer n) and only has two columns. Since it occupies less space, it is possible to fit a bigger portion of it in the primary space, which we have fast access to.

Clustering Index

- Since records are ordered, insertion and deletion cause problems
- To overcome the problem of insertion:
 - Reserve blocks for each distinct value.

Figure 14.3
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.



Secondary Index

- A secondary index provides a secondary means of accessing a file for which some primary access already exists.
- The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- Includes one entry *for each record* in the data file; hence, it is a *dense index*

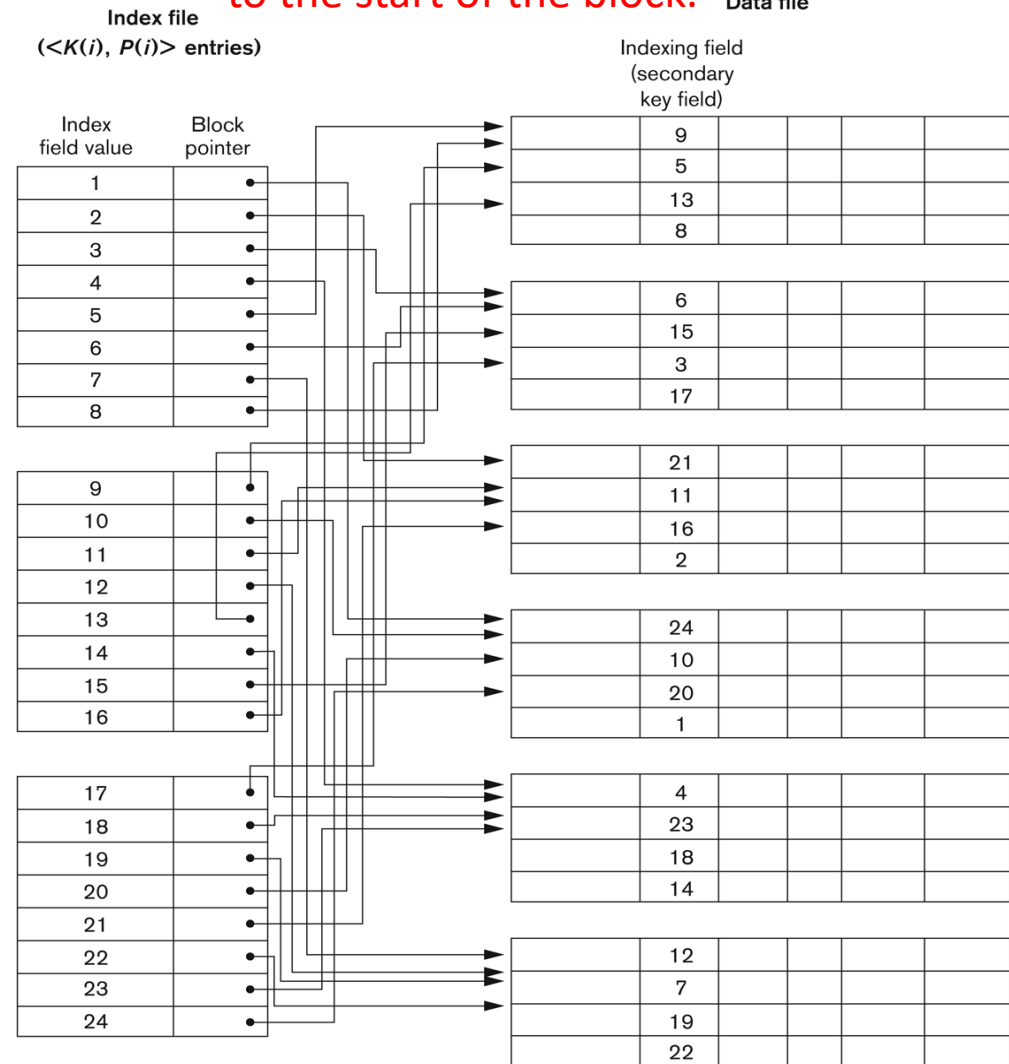
Secondary Index on Key

- Defined on a file for which some primary access already exists.
- A **secondary index on key field** is an ordered file with 2-field entries $\langle K(i), P(i) \rangle$
 - $K(i)$: is the field
 - $P(i)$: is a pointer to a record
- A secondary index must be a **dense** index

Arrows should point to the start of the block.

Figure 14.4

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



Secondary Index on Non-key

- Defined on a file for which some primary access already exists.
- A **secondary index on non-key field** is a two-level ordered file with 2-field entries $\langle K(i), P(i) \rangle$
 - $K(i)$: is the field
 - $P(i)$: is a pointer to a bucket
- A secondary index must be a **dense** index

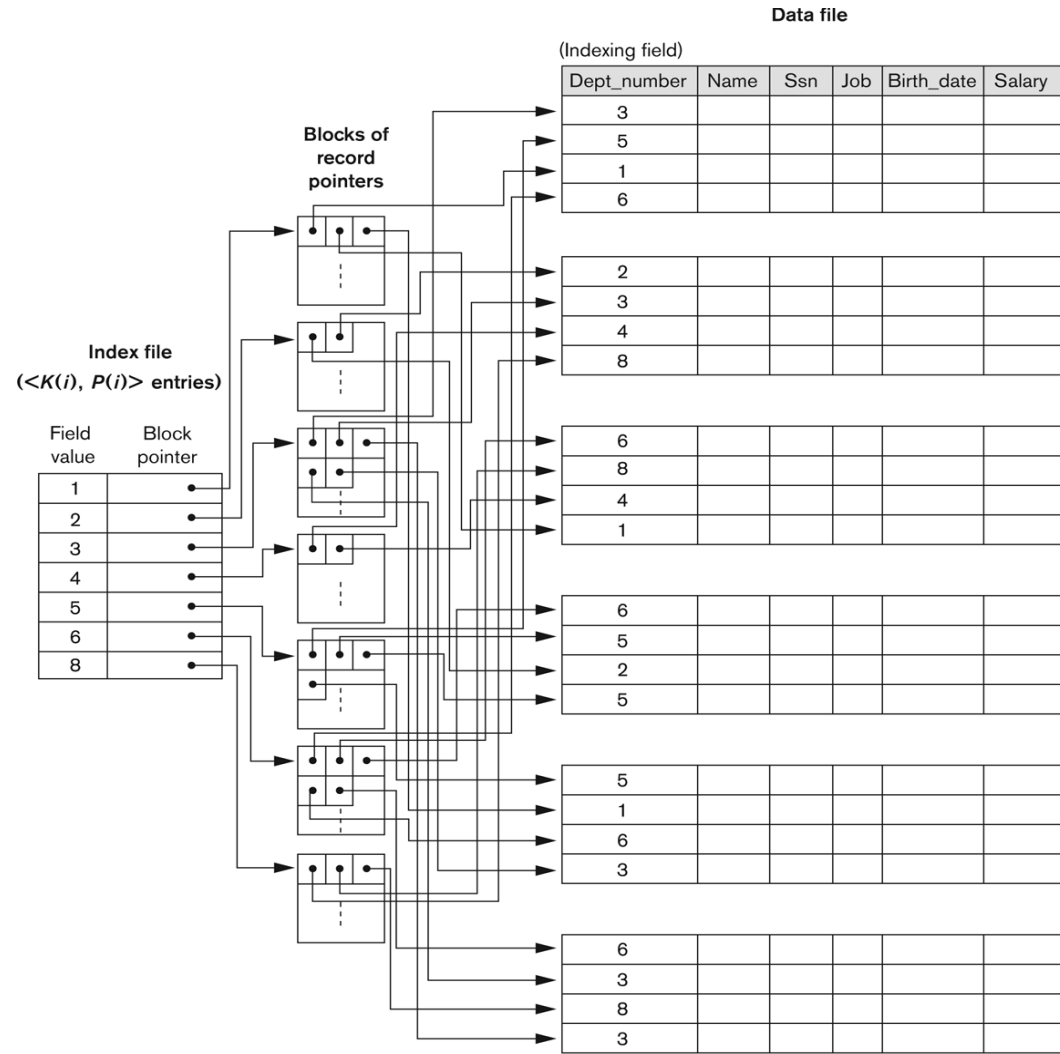


Figure 14.5

A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

Properties of Index Types

Type of Index	Number of (First-level) Index Entries	Dense or Nondense (Sparse)
Primary	Number of blocks in data file	Nondense
Clustering	Number of distinct index field values	Nondense
Secondary (key)	Number of records in data file	Dense
Secondary (nonkey)	Number of records ^b or number of distinct index field values ^c	Dense

Intro to Indexing

Single-level Indexes

Multi-level Indexes

Indexes on Multiple Keys

Multi-Level Indexes

- Because a single-level index is an ordered file:
 - create a primary index to the index itself!
 - original index file is called first-level index
 - index to index is called the second-level index
- Repeat the process until all entries of the top level fit in one disk block
- A multi-level index can be used on any type of first-level index: primary, secondary, clustering

A Two-level Primary Index

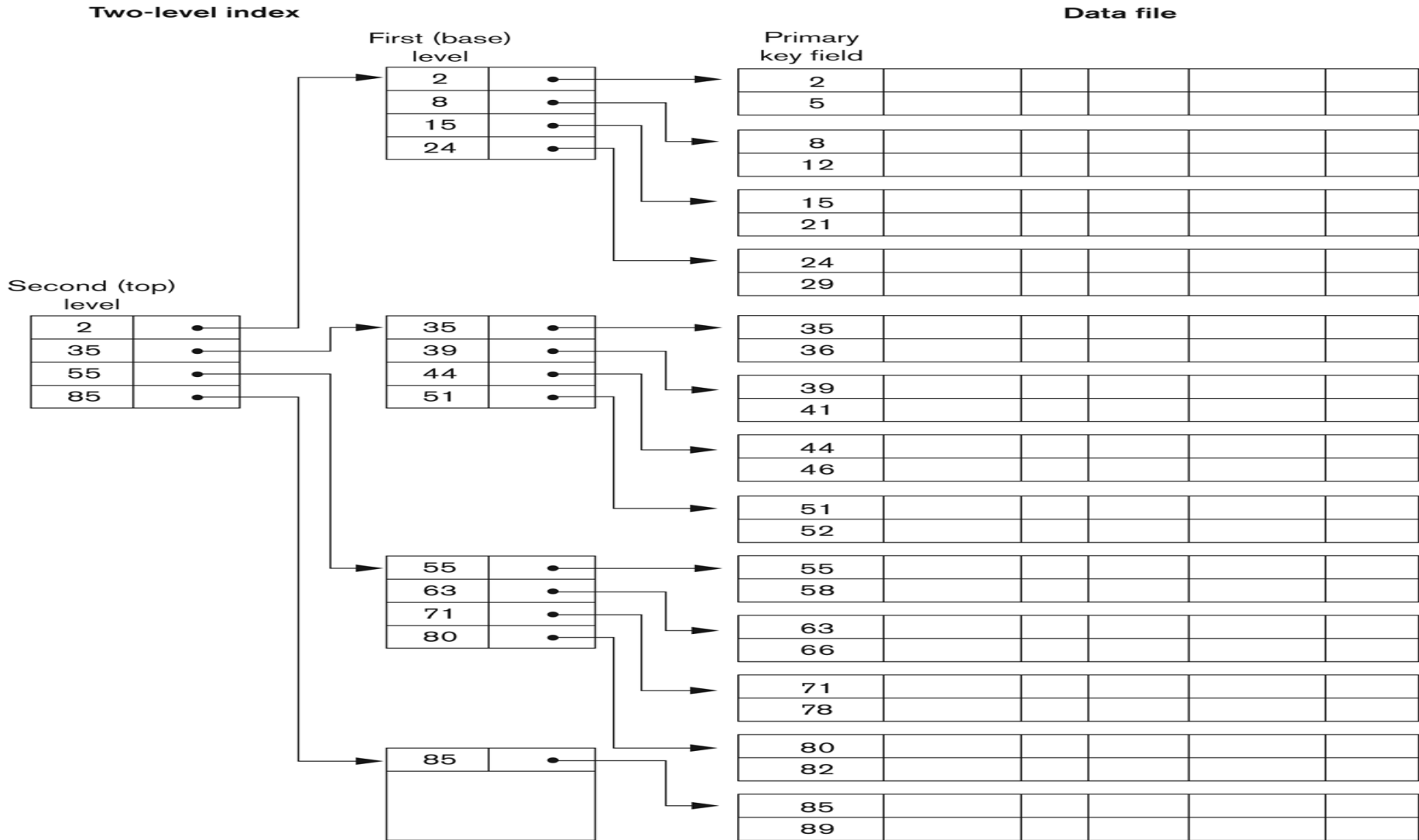


Figure 14.6

A two-level primary index resembling ISAM (Index Sequential Access Method) organization.

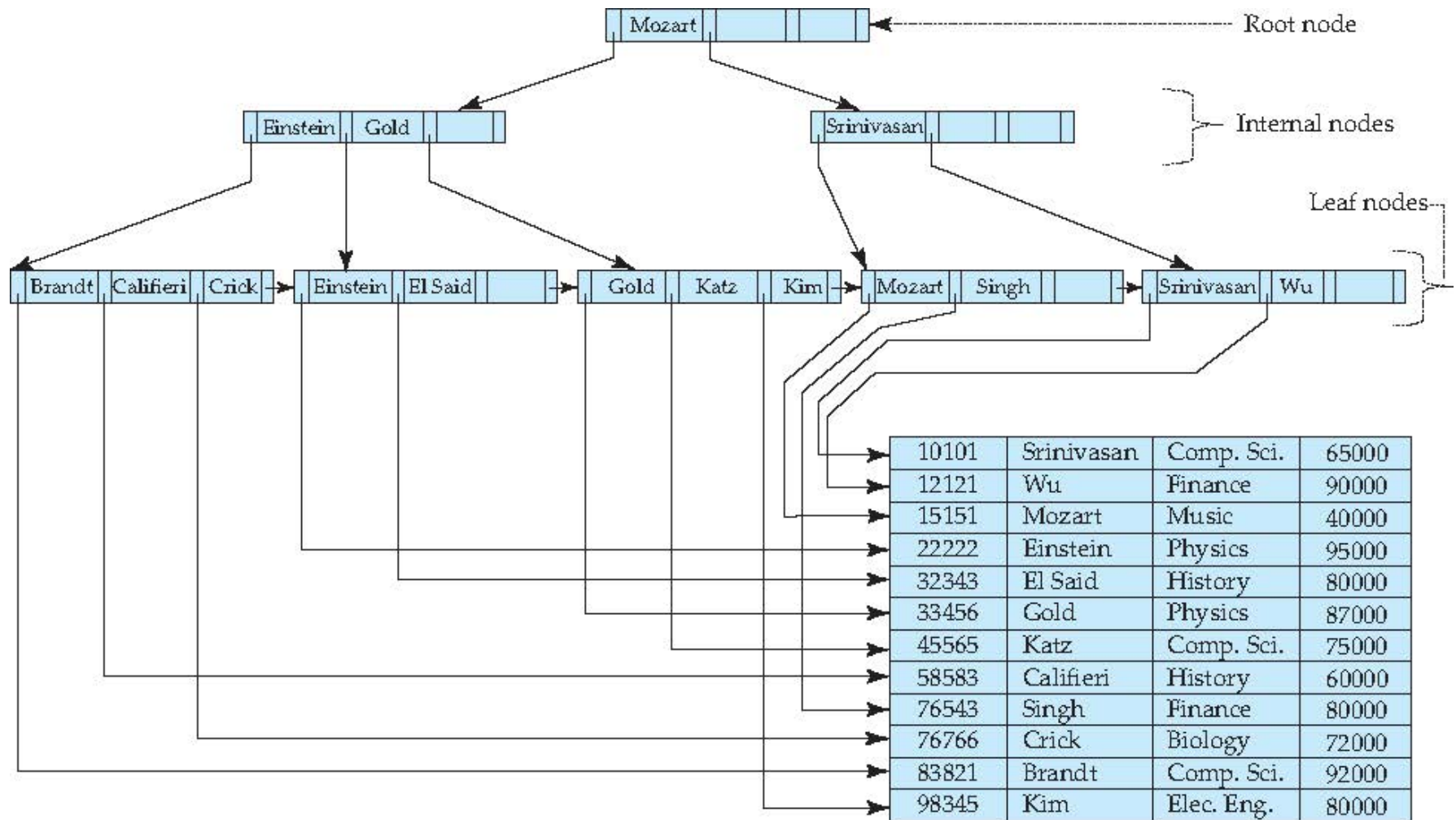
Tree-structured Indexes

- A tree-structured index is one where the search keys are organized into a tree structure.
- Main operations are find, insert, and delete
- Tree-structured indexing techniques support both *range searches* and *equality searches*.

B+ Trees

- B+ Trees are the most widely used index in databases.
- B+ Trees are based on B-Trees, which are a generalization over Binary Search Trees. B+ trees are self balancing with logarithmic ($O(\log n)$) running time for Search, Insert, or Delete operations.
- Advantages
 - Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
 - Reorganization of entire file is not required to maintain performance.
- Disadvantage
 - Extra insertion and deletion overhead
 - Space overhead.

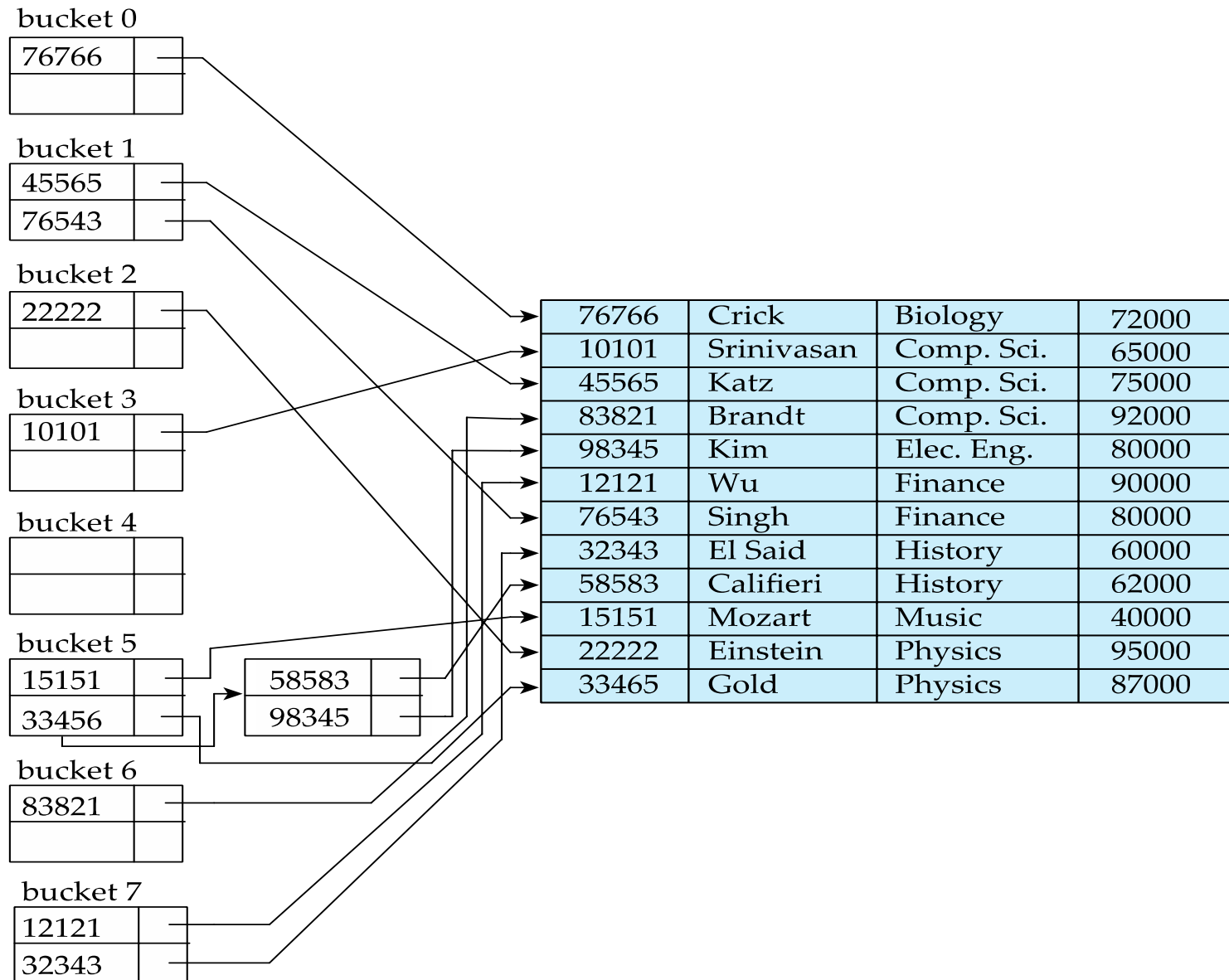
B+ Tree Example



Hashed-based Indexes

- A **hash index** organizes the search keys, with their associated record pointers, into a hash file structure.
- *Hash-based* indexes are usually the best choice for *equality* selections.
 - No traversal of trees
 - Direct computation of where the record must be.
- Hash-based indexes cannot support range searches efficiently.

Hashed-based Index Example



Intro to Indexing

Single-level Indexes

Multi-level Indexes

Indexes on Multiple Keys

Indexes on Multiple Keys

- Assume that we regularly need to run the following query:

```
Select *  
from Movie  
where MovieID>10 and year <1940
```

- Search strategies:
 - Assuming MovieID has an index, but year doesn't: Find records that satisfy the MovieID condition and then select those satisfying the year condition.
 - Assuming year has an index, but MovieID doesn't: Find records that satisfy the year condition and then select those satisfying the MovieID condition.
 - Assuming both have indexes: Find records from each and then iterate through the smaller list and select those exist in the other list.
- What if we had a joint index on MovieID and year?

Indexes on Multiple Keys

EMPLOYEE (Ssn , Dno, Age, Street, City, Zip_code, Salary)

- Consider the query: List the employees in department number 4 whose age is 59.
- Search strategies:
 - Assuming Dno has an index, but Age doesn't: Find records that satisfy the Dno condition and then select those satisfying age condition.
 - Assuming age has an index, but Dno doesn't: Find records that satisfy the Age condition and then select those satisfying the Dno condition.
 - Assuming both have indexes: Find records from each and then iterate through the smaller list and select those exist in the other list.
- What if we had a joint index on both Dno and Age?

Indexes on Multiple Keys

- **Composite search keys:** each key containing more than one attribute (E.g. $\langle \text{Dno}, \text{Age} \rangle$)
- **Ordered index on multiple attributes:** A lexicographic ordering of tuple values on the composite search key

Example

$\langle 3, 20 \rangle$ precedes $\langle 4, 25 \rangle$

Indexes on Multiple Keys

- **Partitioned Hashing:** a key consisting of n components, the hash function is designed to produce a result with n separate hash addresses.

Example:

Suppose Dno and Age are hashed into a 3-bit and 5-bit address

Dno = 4 \rightarrow hash address '100'

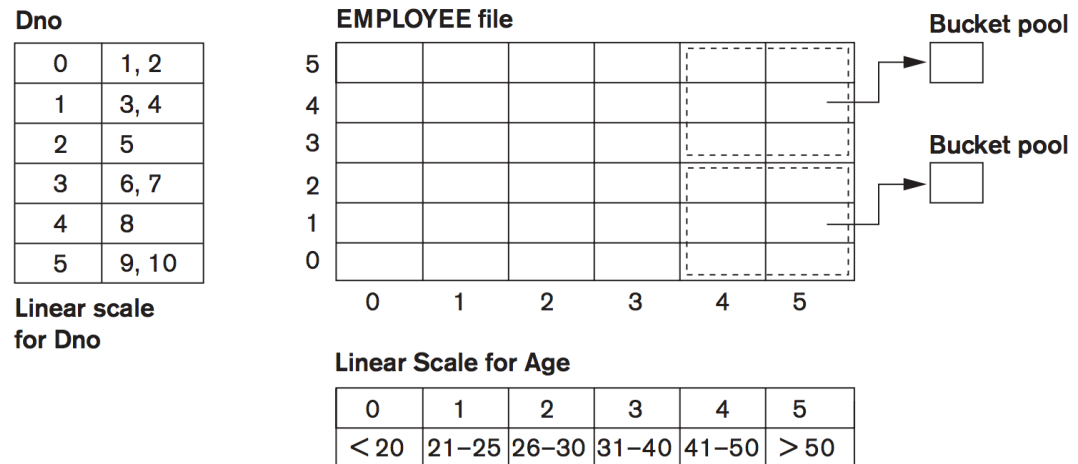
Age = 59 \rightarrow hash address '10101'

searching for Dno = 4 and Age = 59 \rightarrow 100 10101

searching for Age = 59 \rightarrow XXX 10101

Indexes on Multiple Keys

- **Grid Files:** a grid array with one linear scale (or dimension) for each of the search attributes. This method is particularly useful for range queries that would map into a set of cells corresponding to a group of values along the linear scales.



Example:

Dno = 4 and Age = 59 maps into the cell (1, 5)

Dno = 4 maps into row 1

In-class exercise

- Suppose you know that the following queries are the most common queries in the database and that all are roughly equivalent in frequency and importance:

- Find students with $\text{snum} < 1000$
- find students in 'computer science'
- find students that are younger than 20
- Find students in 'computer science' that are '20'

Student(snum,sname,major,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)

- Given this information, decide which attributes should be indexed and whether each index should be a clustered index or an unclustered index. Assume that both B+ trees and hashed indexes are supported by the DBMS and that both single- and multiple-attribute index search keys are permitted.

Indexing

- Suppose you know that the following queries are the most common queries in the database and that all are roughly equivalent in frequency and importance:

Student(snum,sname,major,age)
Class(name,meets_at,room,fid)
Enrolled(snum,cname)

- Find students with $snum < 1000$

This is a range query on primary key, so a single-attribute primary index using B+ Tree indexes

- find students in 'computer science'

This is a equality query on non-primary key, so a single-attribute secondary index using hash-based indexes

- find students that are younger than 20

This is a range query on non-primary key, so a single-attribute secondary index using B+ Tree indexes

- Find students in 'computer science' that are '20'

This is a query on conjunctive selection constraints, so a non-primary key, so a multi-attribute index using partition hashing or grid files.

CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name  
  [index_type]  
  ON tbl_name (index_col_name,...)  
  [index_option]  
  [algorithm_option | lock_option] ...
```

index_type:
 USING {BTREE | HASH}

algorithm_option:
 ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
 LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

<https://dev.mysql.com/doc/refman/5.7/en/create-index.html>

Learning Objectives Revisited

Description	Tag
Explain the benefits of using indexes in database systems.	Indexing-benefits
Describe the notion of a dense index.	Dense-index
Describe the notion of a sparse index.	Sparse-index
Describe the properties of primary indexes.	Primary-index
Describe the properties of clustering indexes.	Clustering-index
Describe the properties of secondary indexes.	Secondary-index
Describe the properties of multi-level indexes.	Multi-level-index
Describe the properties of tree-structured indexes.	Tree-index
Explain how a tree-structured index supports both range and point queries.	Tree-index-property
Describe the properties of hash-based indexes.	Hash-index
Explain why hash-based index only support point queries.	Hash-index-property
Compare and contrast the performance of different indexes.	Compare-Indexes
Given information about the most common queries in a database, recommend which attributes using which methods should be indexed.	Recommend-indexes