COL362/632 Introduction to Database Management Systems

Database Systems – Indexing

Kaustubh Beedkar

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

Motivation

Contents

Motivation

Contents

Index

Motivation

Contents

PART FIVE ■ STORAGE MANAGEMENT AND INDEXING

Chapter 12 Physical Storage Systems

Chapter 13 Data Storage Structures

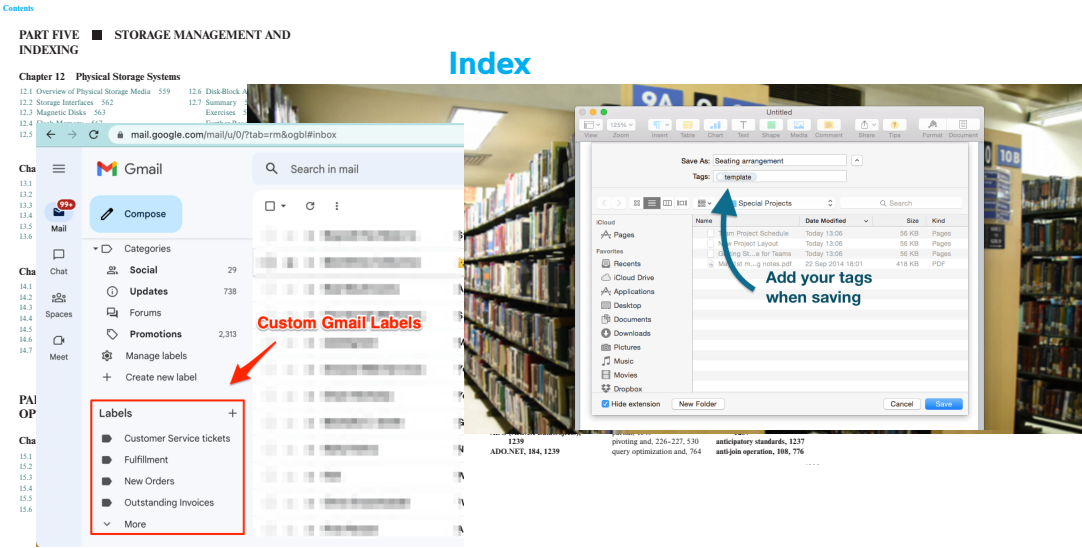
Chapter 14 Indexing

PART SIX ■ QUERY PROCESSING AN OPTIMIZATION

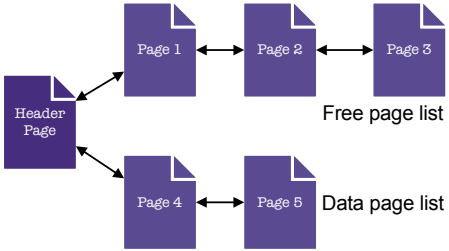
Chapter 15 Query Processing

Index

Motivation

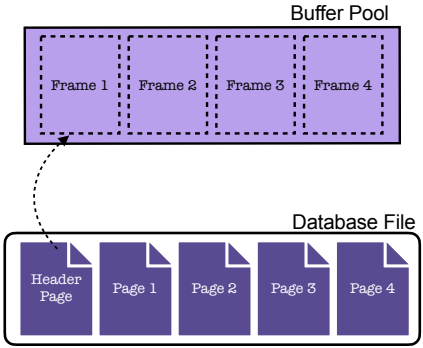

Contents
PART FIVE ■ STORAGE MANAGEMENT AND INDEXING
Chapter 12 Physical Storage Systems
Index
mail.google.com/mail/u/0/?tab=rm&ogbl#inbox
Gmail
Search in mail
Compose
Mail
Chat
Spaces
Meet
Categories
Social 29
Updates 738
Forums
Promotions 2,313
Manage labels
Create new label
Custom Gmail Labels
Labels
Customer Service tickets
Fulfillment
New Orders
Outstanding Invoices
More
Untitled
Save As: Seating arrangement
Tags: template
Special Projects
Add your tags when saving
Pages
Recents
iCloud Drive
Applications
Desktop
Documents
Downloads
Pictures
Music
Movies
Dropbox
Hide extension
New Folder
Cancel
Save
1239
ADO.NET, 184, 1239
pivoting and, 226–227, 530
query optimization and, 764
anticipatory standards, 1237
anti-join operation, 108, 776

Accessing Records in Databases

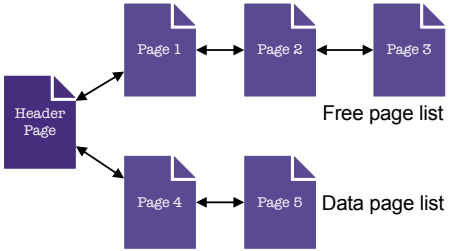


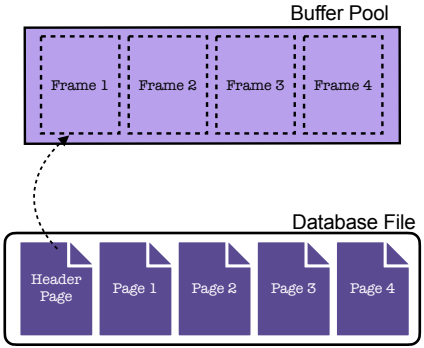
Recall: Heap File Organization

- Good for scanning all records



Accessing Records in Databases

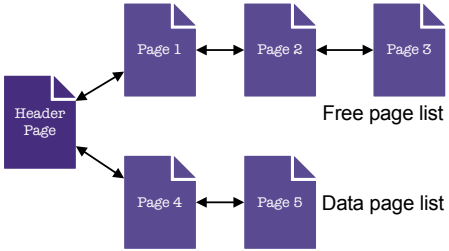


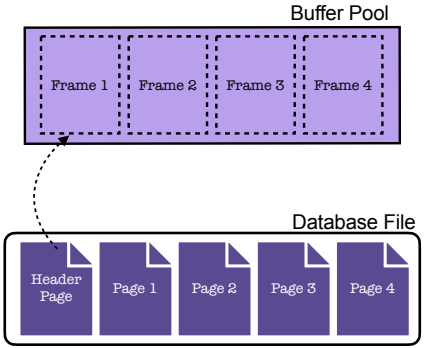


Recall: Heap File Organization

- Good for scanning all records
- But, what if we are looking for "certain" records
  - `select * from Order where return_status = 'I'`
  - `select name from Product where price < 100`

Accessing Records in Databases



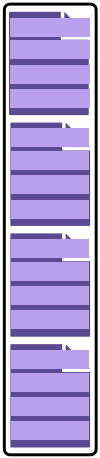


Recall: Heap File Organization

- Good for scanning all records
- But, what if we are looking for "certain" records
  - `select * from Order where return_status = 'I'`
  - `select name from Product where price < 100`
- How to **optimally** retrieve records?

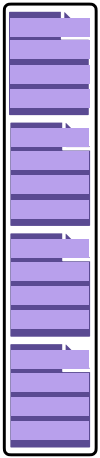
Accessing Records in Database

Accessing Records in Database



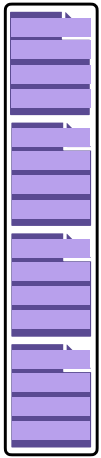
Accessing Records in Database

- Assume block is b (KB), each record is r (KB), and n records



Accessing Records in Database

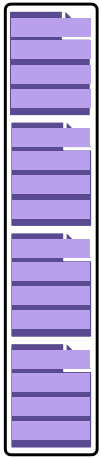
- Assume block is b (KB), each record is r (KB), and n records
- $O(\frac{nr}{b})$ for sequential scan



Accessing Records in Database

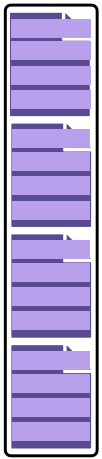
- Assume block is b (KB), each record is r (KB), and n records
- $O(\frac{nr}{b})$ for sequential scan

- Improve access by
  - Keeping sorted files
  - $O(\log_2 \frac{nr}{b})$ for binary search



Accessing Records in Database

- Assume block is b (KB), each record is r (KB), and n records
- $O(\frac{nr}{b})$ for sequential scan

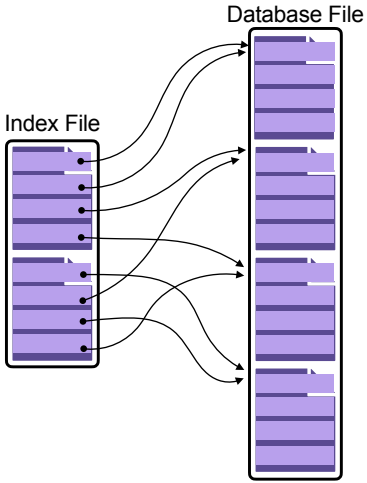


- Improve access by
  - Keeping sorted files
  - $O(\log_2 \frac{nr}{b})$ for binary search
  - **Using Indexes!**
  - B+ trees
  - Hash Index
  - Bit Maps
  - Bloom Filters
  - ...

Indexing - basics

Index

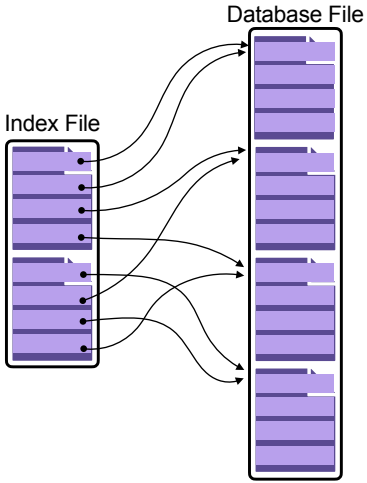
- Data structure that organizes records of a file
- Enables efficiently searching records using **search key**



Indexing - basics

Index

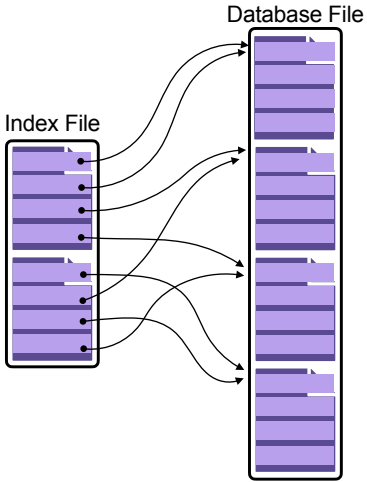
- Data structure that organizes records of a file
- Enables efficiently searching records using **search key**
- **Note:** Index is also a database file!



Indexing - basics

Index

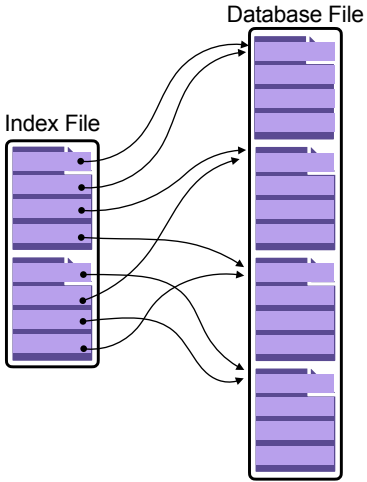
- Data structure that organizes records of a file
- Enables efficiently searching records using **search key**
- **Note:** Index is also a database file!
- Is a collection of **data entries**
  - An entry in index is different from record
  - But has information to locate records



Indexing - basics

Index

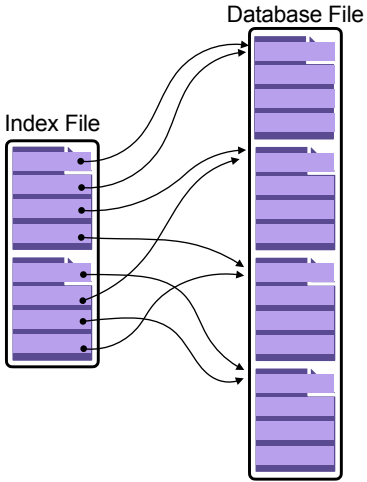
- Data structure that organizes records of a file
- Enables efficiently searching records using **search key**
- **Note:** Index is also a database file!
- Is a collection of **data entries**
  - An entry in index is different from record
  - But has information to locate records
- Two basic types
  1. Ordered Index – Search keys are stored in sorted order
  2. Hash Index – Search keys are distributed uniformly across *buckets* using a hash function



Indexing - basics

Index

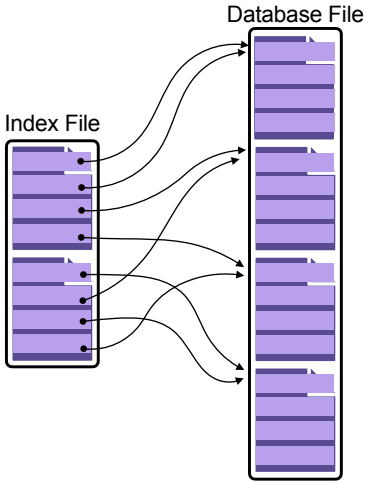
- Data structure that organizes records of a file
- Enables efficiently searching records using **search key**
- **Note:** Index is also a database file!
- Is a collection of **data entries**
  - An entry in index is different from record
  - But has information to locate records
- Two basic types
  1. Ordered Index – Search keys are stored in sorted order
  2. Hash Index – Search keys are distributed uniformly across *buckets* using a hash function
- A database file can have many indices



Indexing - basics

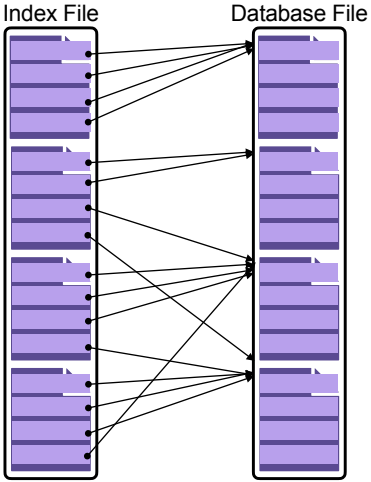
Index

- Data structure that organizes records of a file
- Enables efficiently searching records using **search key**
- **Note:** Index is also a database file!
- Is a collection of **data entries**
  - An entry in index is different from record
  - But has information to locate records
- Two basic types
  1. Ordered Index – Search keys are stored in sorted order
  2. Hash Index – Search keys are distributed uniformly across *buckets* using a hash function
- A database file can have many indices
- Updating file also requires updating the index



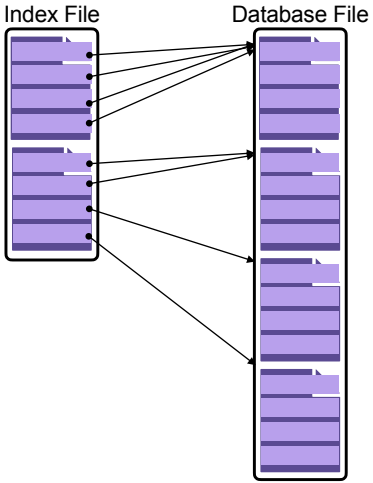
Dense Index

- Data entries based on sorted ordering of values
- There is an entry in the index **for every** search-key value in the file



Sparse Index

- Data entries based on sorted ordering of values
- There is an entry in the index **for only some** search-key value in the file
- **Note** Can only be used if the table is stored in sorted order of the search key



Primary Index and Secondary Index

Primary Index

- Search key contains the primary key
- No duplicate search keys
- A table can have only one primary index

Secondary Index

- Any index other than primary index

Note: Index on candiate key is called a **unique index**

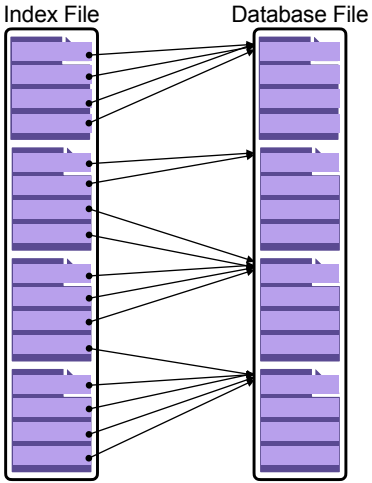
Clustered Index

- Search key also defines the sequential order of the file
- Search key often the primary key
- Can be both dense and sparse

Dense Clustered Index

- Data entry only points to block containing the first record

Note: Sparse Index $\implies$ Clustered Index



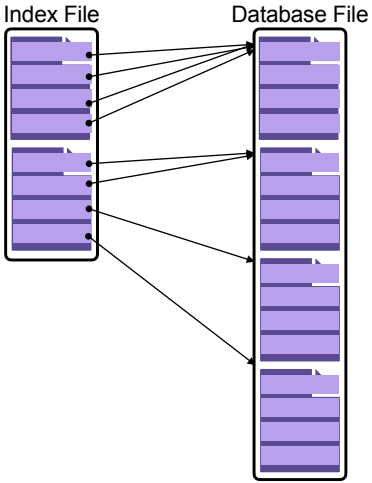
Clustered Index

- Search key also defines the sequential order of the file
- Search key often the primary key
- Can be both dense and sparse

Dense Clustered Index

- Data entry only points to block containing the first record

Note: Sparse Index $\implies$ Clustered Index

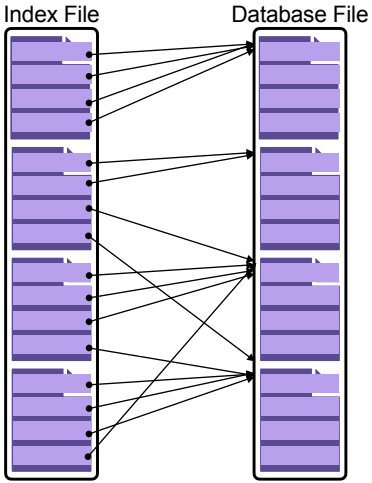


Non-clustered Index

- Search key specifies an order different from the sequential order of the file

Dense Non-clustered Index

- Data entry must store block pointers to all records with the same search key value

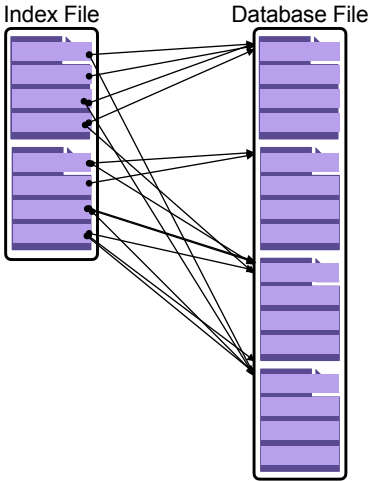


Non-clustered Index

- Search key specifies an order different from the sequential order of the file

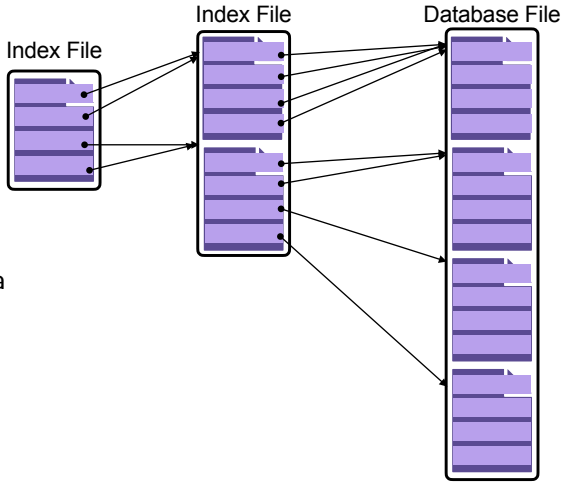
Dense Non-clustered Index

- Data entry must store block pointers to all records with the same search key value

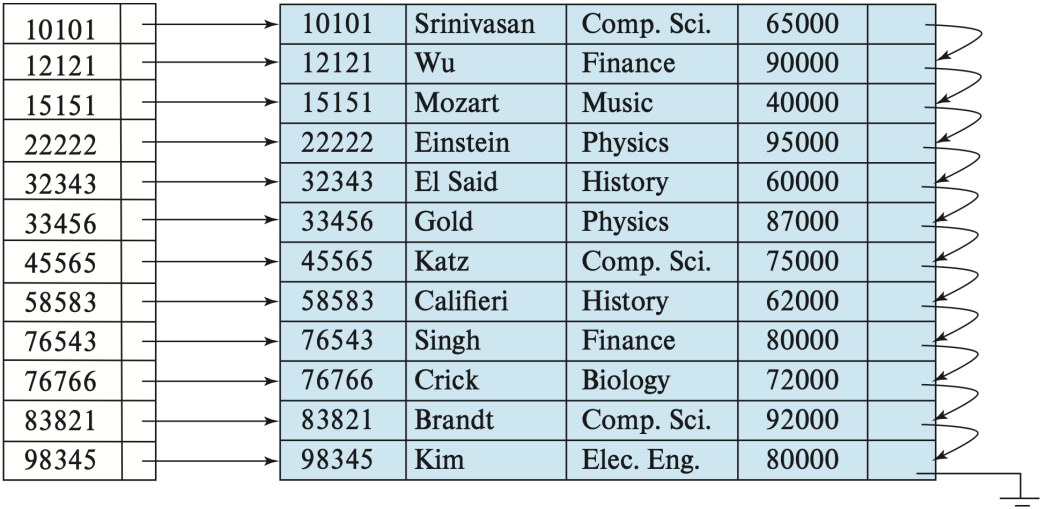


Multi-level Index

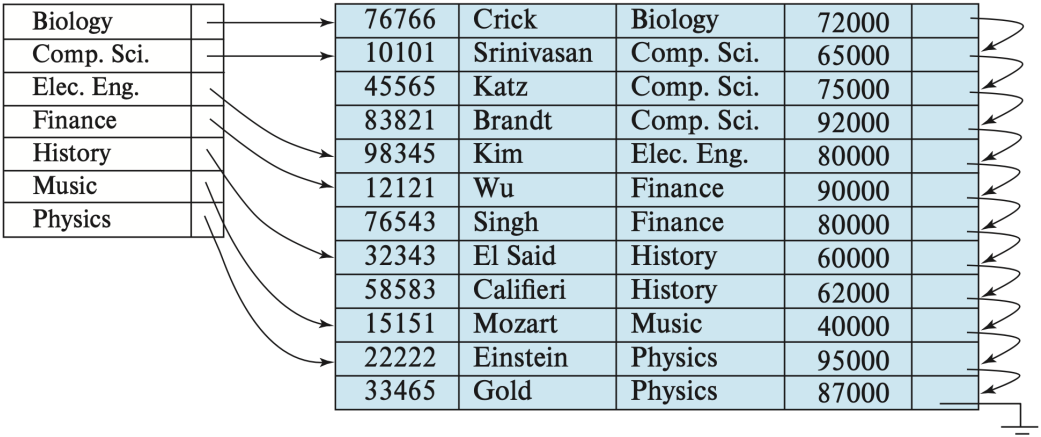
- Sometimes, index may not fit in memory
- Treat index as sequential file and create a **sparse outer** index
- Can be repeated



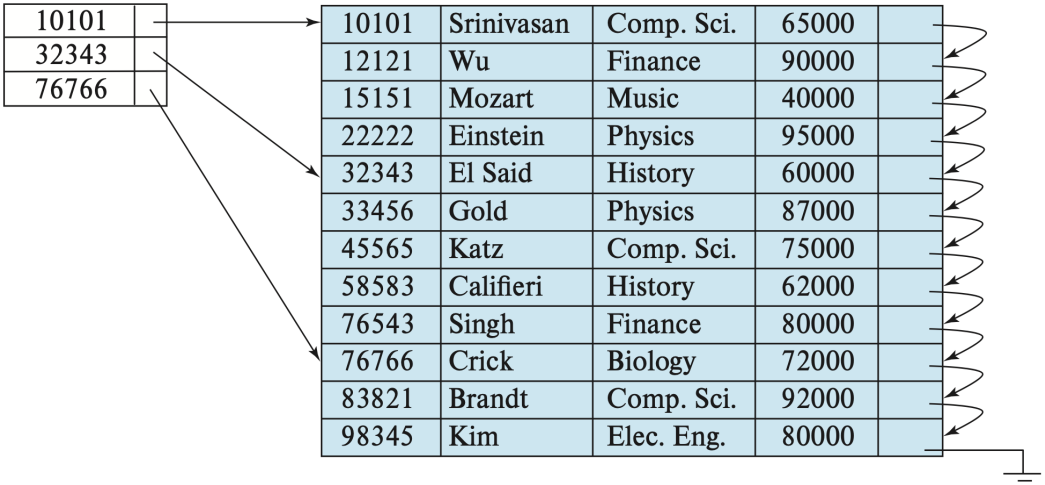
Examples (from Book B1)



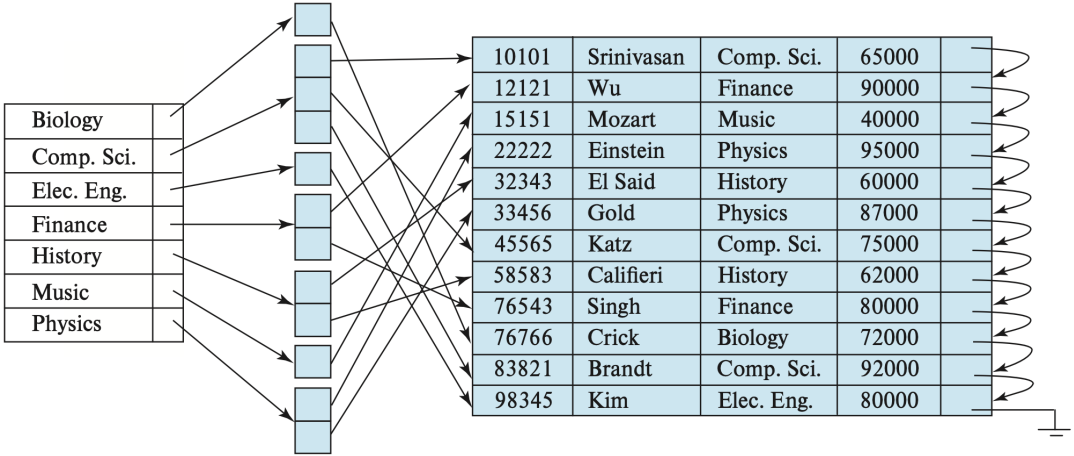
Examples (from Book B1)



Examples (from Book B1)



Examples (from Book B1)



Updating Index – Insertions

Dense Index

- If search key !in index
  add a data entry at appropriate position
- Else If non-clustering index, add pointer to data entry
- Else place the **record** appropriately

Sparse Index

- If new block, add new data entry
- If new record has least search-key value, update the data entry!

Updating Index – Deletions

Dense Index

- It it was the only record, delete the index entry
- Else, for non-clustering index, delete the pointer
- Else, update the data entry by pointing the next record

Sparse Index

- If the index does not contain data entry with search key value, do nothing; Else,
- If it was the only record, update the index entry with next search key value
- If next search key value present in index, delete the index entry
- Else, if index entry points to recorded being deleted, update the index to point to the next record with the same search key value (record can be in another block)

HomeWork

- Find out which is the largest table
- Check if it has index
- create index if there is none for some attribute
- benchmark multiple queries with and without index

Creating/dropping an index in PostgreSql

- `CREATE UNIQUE INDEX title_idx ON films (title)`
- `CREATE INDEX title_idx ON films (title) WITH (deduplicate_items = off)`
- More examples on https://www.postgresql.org/docs/current/sql-createindex.html
- `drop index index_name`

On a side note

Which one is clustered and which is non-clustered?

Contents

PART FIVE ■ STORAGE MANAGEMENT AND INDEXING

Chapter 12 Physical Storage Systems

Chapter 13 Data Storage Structures

Chapter 14 Indexing

PART SIX ■ QUERY PROCESSING AND OPTIMIZATION

Chapter 15 Query Processing

Index