R. C. Patel Institute of Technology, Shirpur

Department of Computer Engineering

Advanced Database Management System(PCC050402T)

by

Dr. Ujwala Patil

Unit I

Advance Databases

Indexing and Hashing

- We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

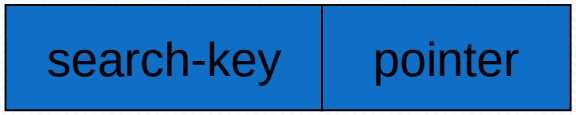
- Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.

- Indexing in database systems is similar to what we see in books.

- The index structures are additional files on disk that provide **secondary access paths**, which provide alternative ways to access the records without affecting the physical placement of records in the primary data file on disk.

- They enable efficient access to records based on the **indexing fields** that are used to construct the index.

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form



- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across “buckets” using a “hash function”.

Index Evaluation Metrics

- Access types supported efficiently.

E.g.,

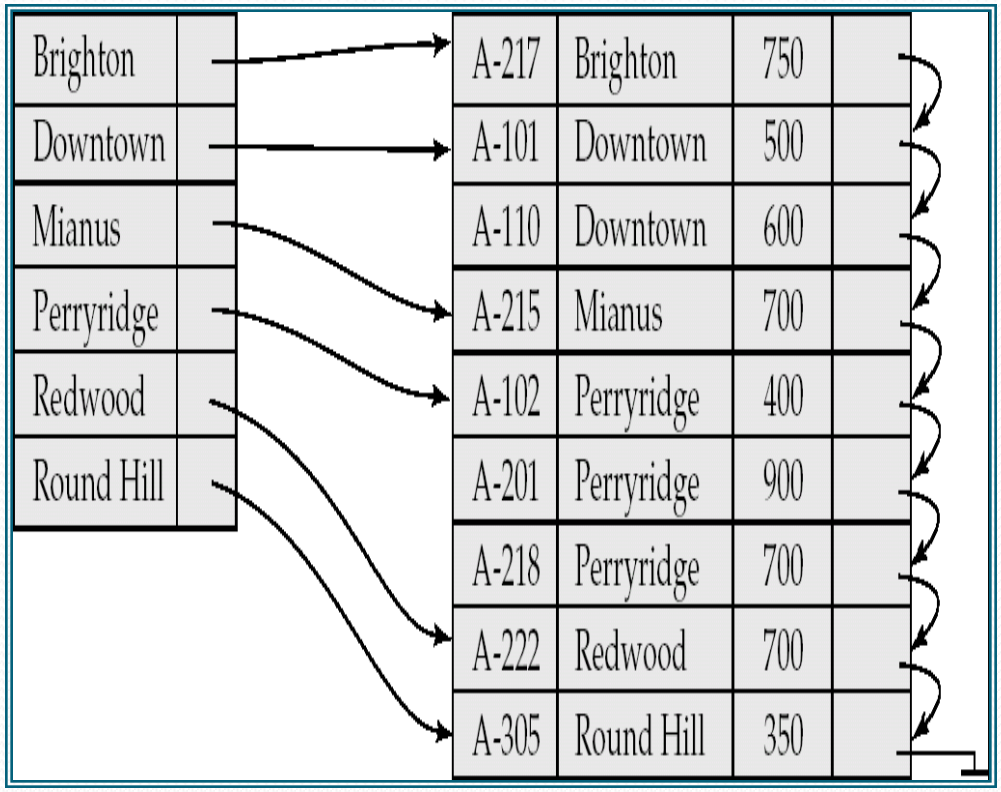
  - records with a specified value in the attribute
  - or records with an attribute value falling in a specified range of values.
- Access time
- Insertion time
- Deletion time
- Space overhead

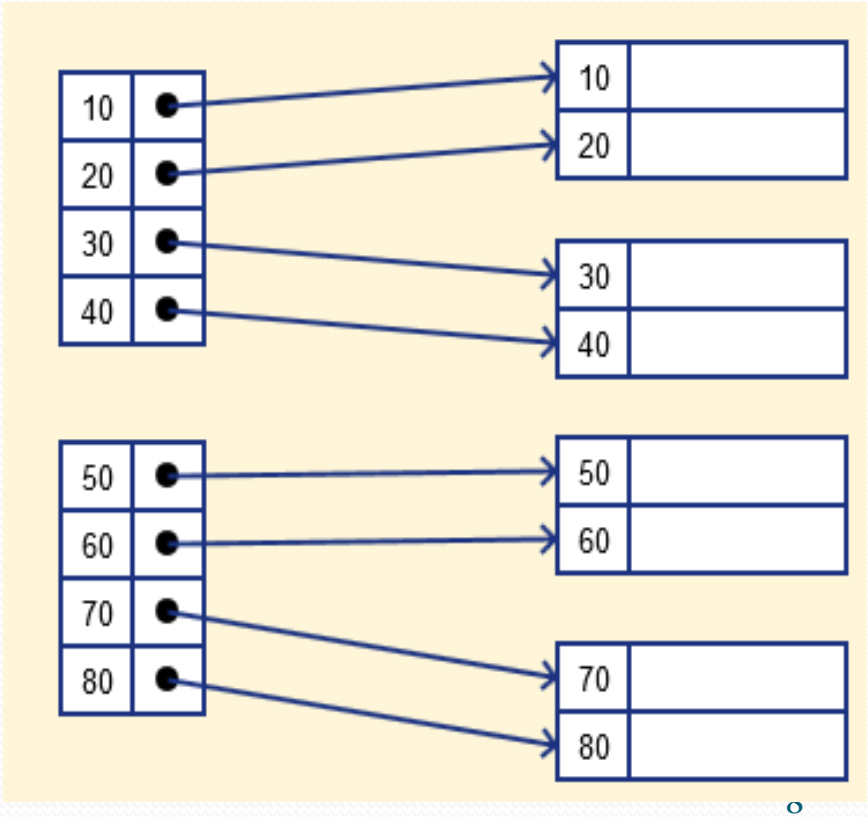
Types

- Classification of Index
  - Dense
  - Sparse
- Types of Index
  - Single Level – (Index and Data block direct communication)
    - Primary Index- (Primary key + ordered)
    - Clustered Index- (Nonkey + ordered)
    - Secondary Index- (Nonkey + unordered)
  - Multi-level - (Index and Data block indirect communication
    - B-Tree
    - B+-Tree

Dense Index Files

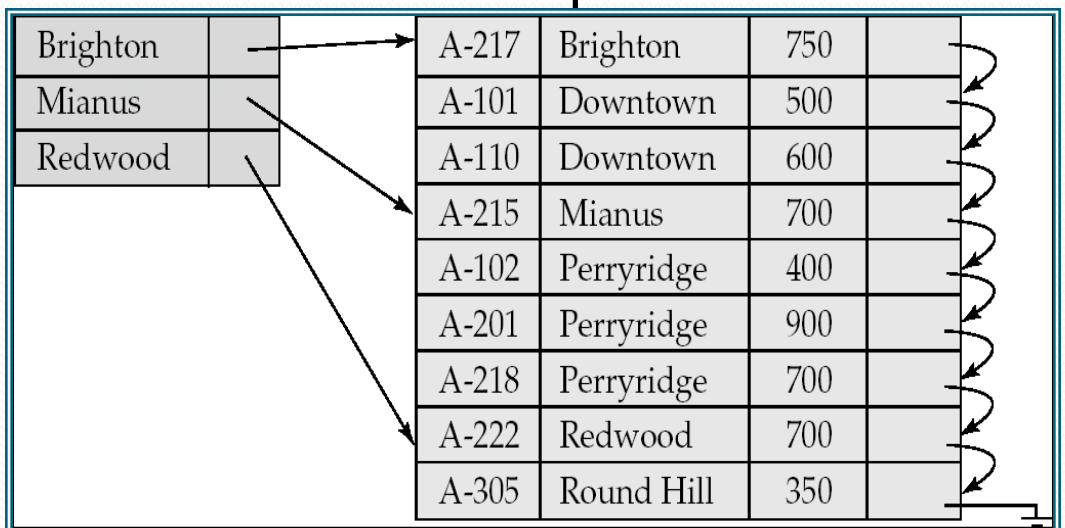
- Dense index — Index record appears for every search-key value in the file.

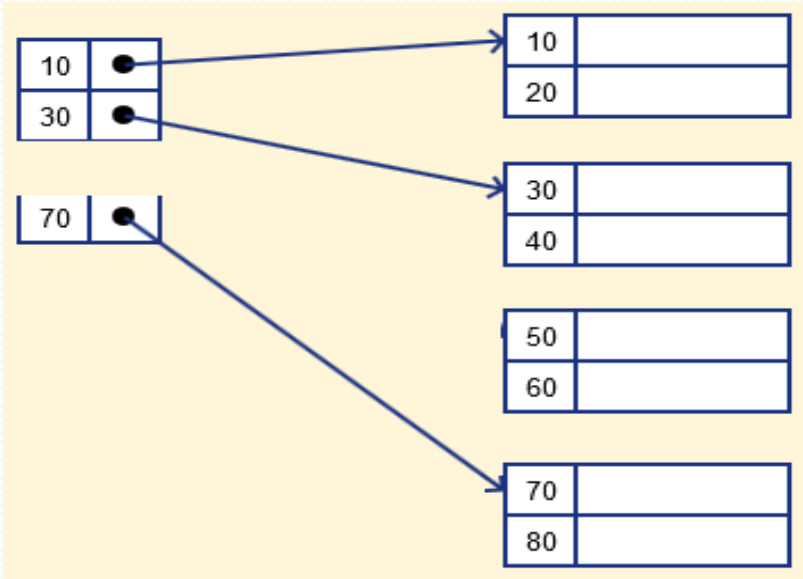




Sparse Index Files

- 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.





Types of Single-level Ordered Indexes

- Indexing field (attribute)
  - Index stores each value of the index field with list of pointers to all disk blocks that contain records with that field value
- Values in index are ordered
- Primary index
  - Specified on the ordering key field of ordered file of records

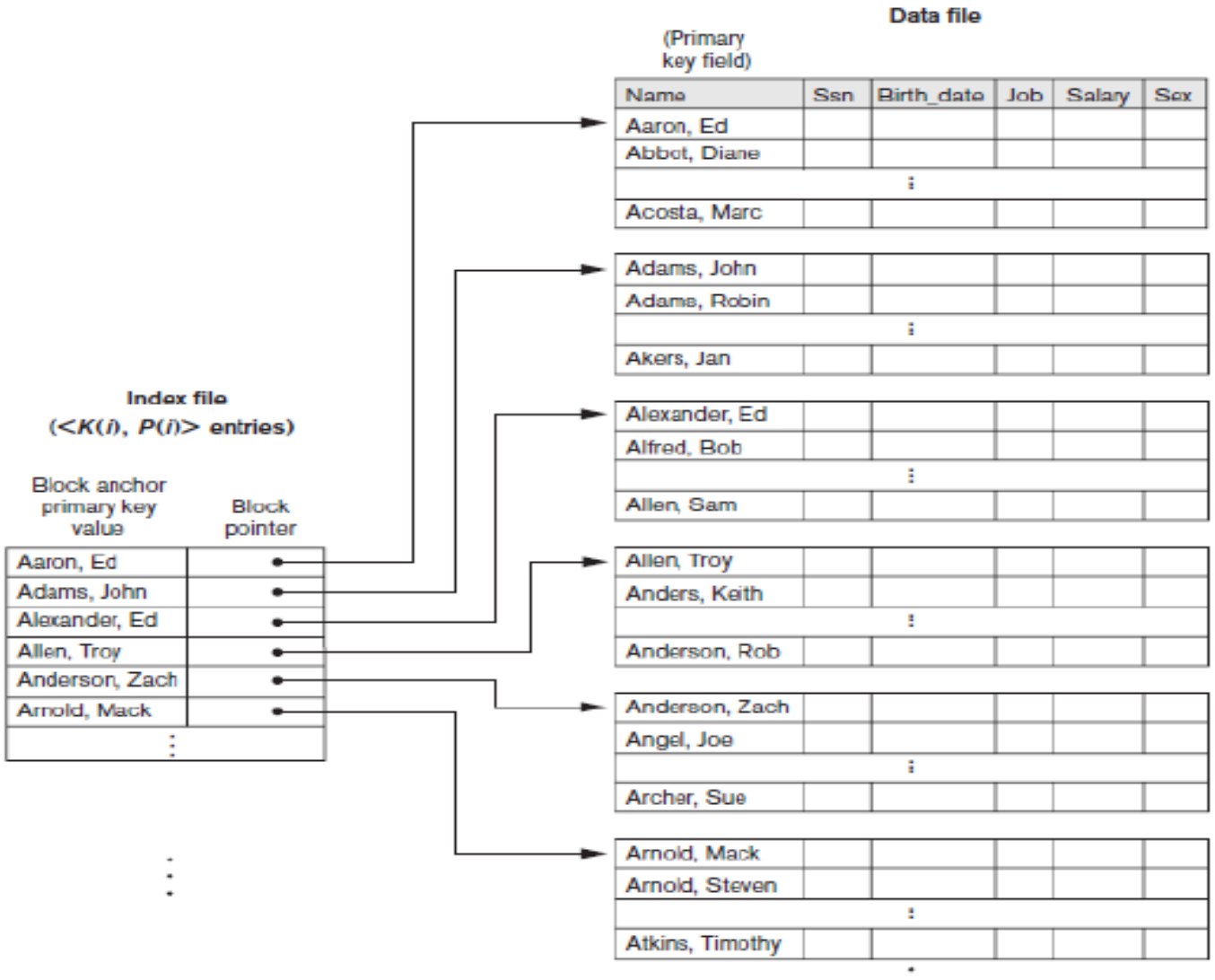
Types of Single-level Ordered Indexes(Contd)

- Clustering index
  - Used if numerous records can have the same value for the ordering field
- Secondary index
  - Can be specified on any nonordering field
  - Data file can have several secondary indexes

Primary Indexes

- A **primary index** is an ordered file whose records are of fixed length with two fields, and it acts like an access structure to efficiently search for and access the data records in a data file.
- The first field is of the same data type as the ordering key field—called the **primary key**—of the data file, and the second field is a pointer to a disk block (a block address).
- There is one **index entry** (or **index record**) in the index file for each *block* in the data file. Each index entry has the value of the primary key field for the *first* record in a block and a pointer to that block as its two field values

Primary Indexes(Contd..)



Primary index on the ordering key field of the file :

Example

- Suppose that we have an ordered file with r = 30,000 records stored on a disk with block size B = 1024 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes. Also, suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file.

- Find out:

  1. Cost of searching for a record using Binary Search on the data file
  2. Cost of searching for a record using the index

- Suppose that we have an ordered file with r = 30,000 records stored on a disk with block size B = 1024 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes. Also, suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file. Find out the cost of searching for a record using Binary Search on the data file

• Blocking factor:

• bfr = $\lfloor (B/R) \rfloor = \lfloor (1024/100) \rfloor = 10$ records per block.

• The number of blocks needed for the file is – b = $\lceil (r/bfr) \rceil = \lceil (30000/10) \rceil = 3000$ blocks.

• A binary search on the data file would need approximately

$$\lceil \log_2 b \rceil = \lceil (\log_2 3000) \rceil = 12 \text{ block accesses}$$

- Suppose that we have an ordered file with r = 30,000 records stored on a disk with block size B = 1024 bytes. File records are of fixed size and are unspanned, with record length R = 100 bytes
- Also, suppose that the ordering key field of the file is V = 9 bytes long, a block pointer is P = 6 bytes long, and we have constructed a primary index for the file.
- Find out the cost of searching for a record using the index

- The size of each index entry : –
- Ri= (9 + 6) = 15 bytes, So the blocking factor for the index is bfri = $\lfloor (B/Ri) \rfloor = \lfloor (1024/15) \rfloor$ = 68 entries per block.
- The total number of index entries is equal to the number of blocks in the data file, which is 3000.
- The number of index blocks is hence bi = $\lceil (ri/bfri) \rceil = \lceil (3000/68) \rceil$ = 45 blocks.
- To perform a binary search on the index file would need $\lceil (log2bi) \rceil = \lceil (log245) \rceil$ = 6 block accesses.
- To search for a record using the index, we need one additional block access to the data file for a total of 6 + 1 = 7 block accesses—an improvement over binary search on the data file, which required 12 disk block accesses.

- Major problem with a primary index—as with any ordered file—is insertion and deletion of records. With a primary index, the problem is compounded because if we attempt to insert a record in its correct position in the data file, we must not only move records to make space for the new record but also change some index entries, since moving records will change the *anchor records* of some blocks.

- **Solutions**
  - Use unordered overflow file

  With this approach, new records are simply inserted at the end of the overflow file instead of in the main data file. This overflow file can then be merged with the main data file periodically. The advantage with this technique is that inserting new record becomes very efficient.

  - Use linked list of overflow record

  As with the previously mentioned overflow file, searching the overflow records involves a linear search, only this time the records to be searched belong to a specific block and not all of the records in the table. This will, on average, increase the efficiency of the search algorithm.

Clustering Indexes

- If file records are physically ordered on a nonkey field—which *does not* have a distinct value for each record—that field is called the **clustering field** and the data file is called a **clustered file.**
- We can create a different type of index, called a **clustering index**, to speed up retrieval of all the records that have the same value for the clustering field.
- This differs from a primary index, which requires that the ordering field of the data file have a *distinct value* for each record.

- A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file, and the second field is a disk block pointer.

- A clustering index is another example of a *nondense* index because it has an entry for every *distinct value* of the indexing field, which is a nonkey by definition and hence has duplicate values rather than a unique value for every record in the file.

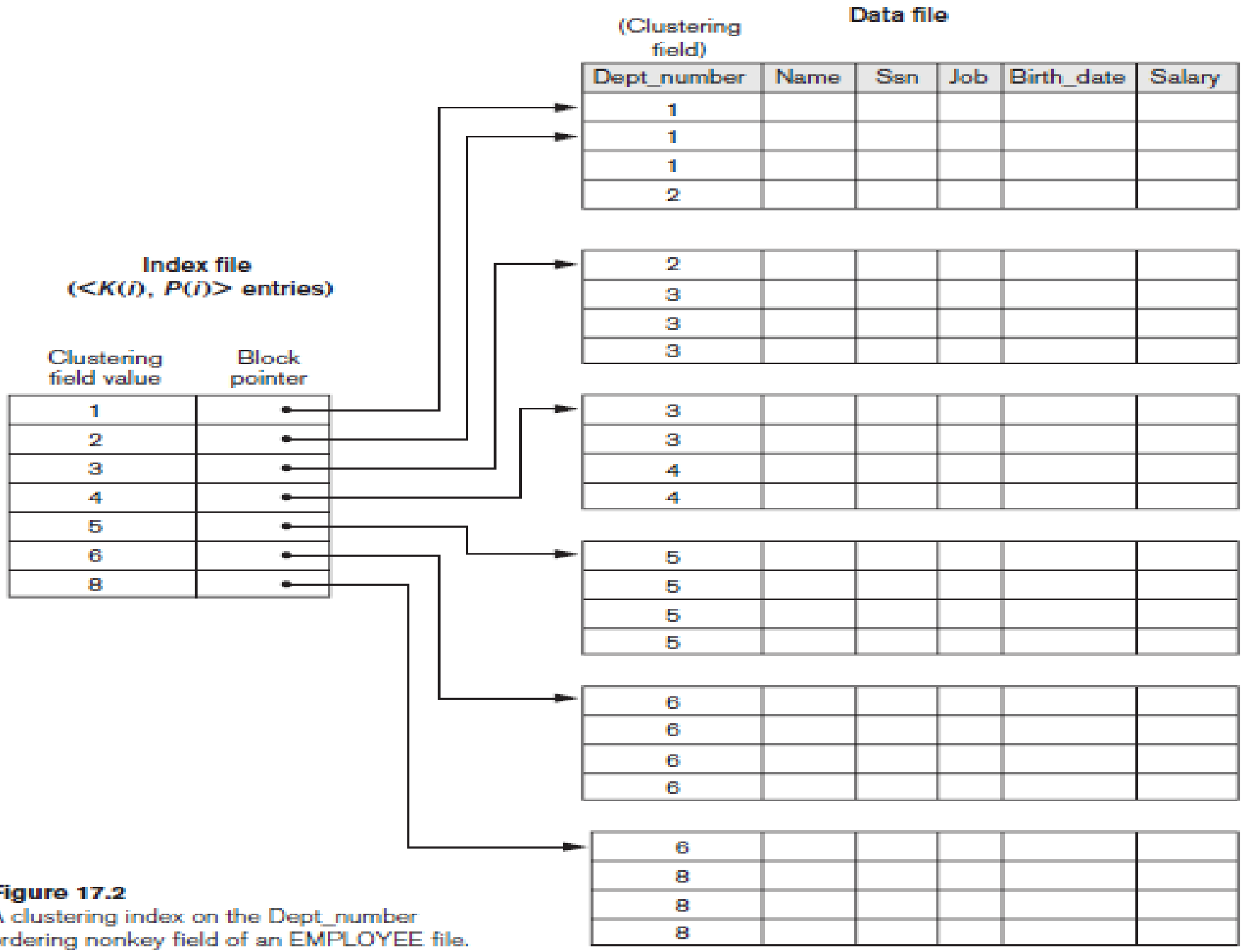


Figure 17.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

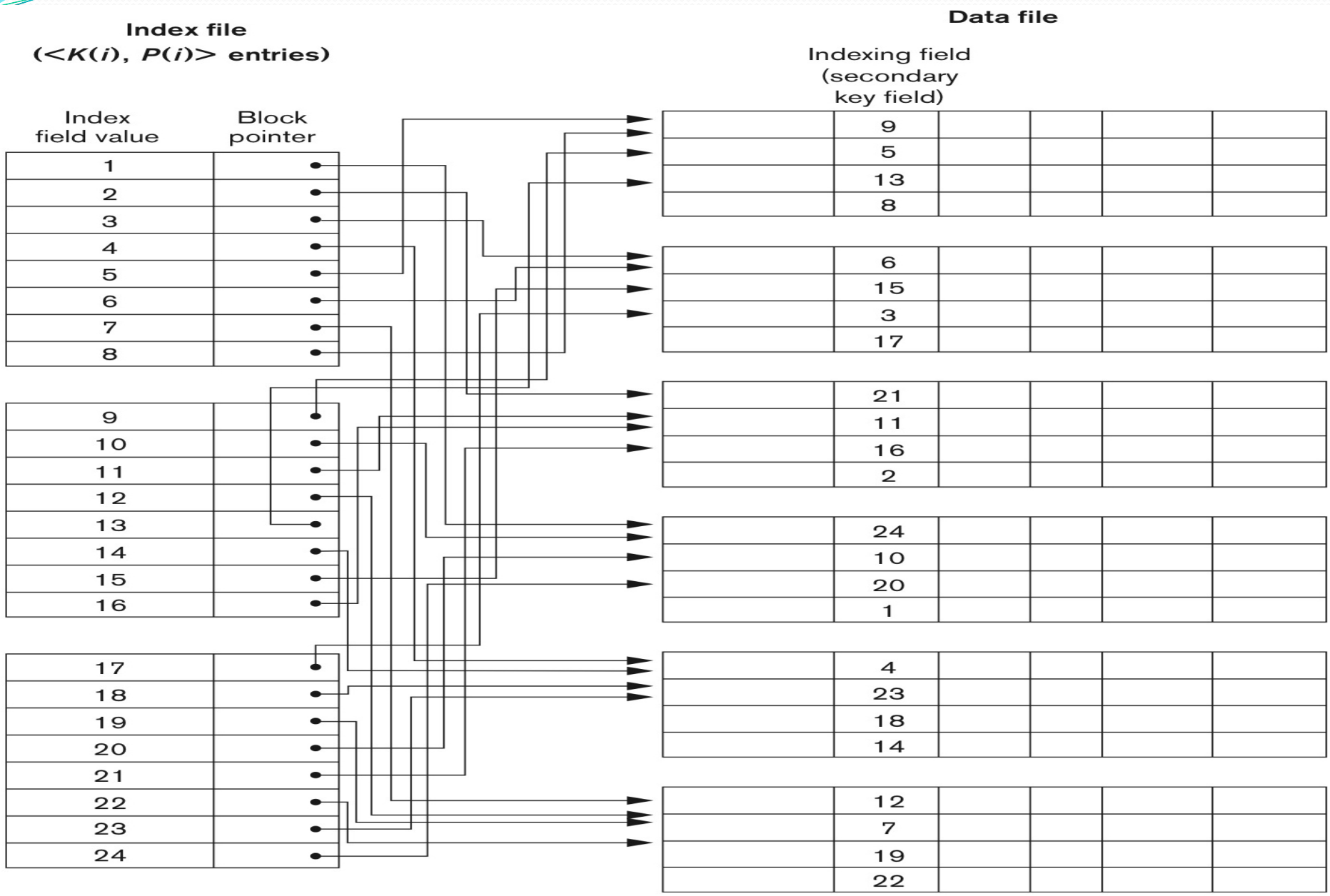
- The record insertion and deletion still cause problems because the data records are physically ordered. To alleviate the problem of insertion, it is common to reserve a whole block (or a cluster of contiguous blocks) for *each value* of the clustering field; all records with that value are placed in the block (or block cluster). This makes insertion and deletion relatively straightforward.

Secondary Indexes

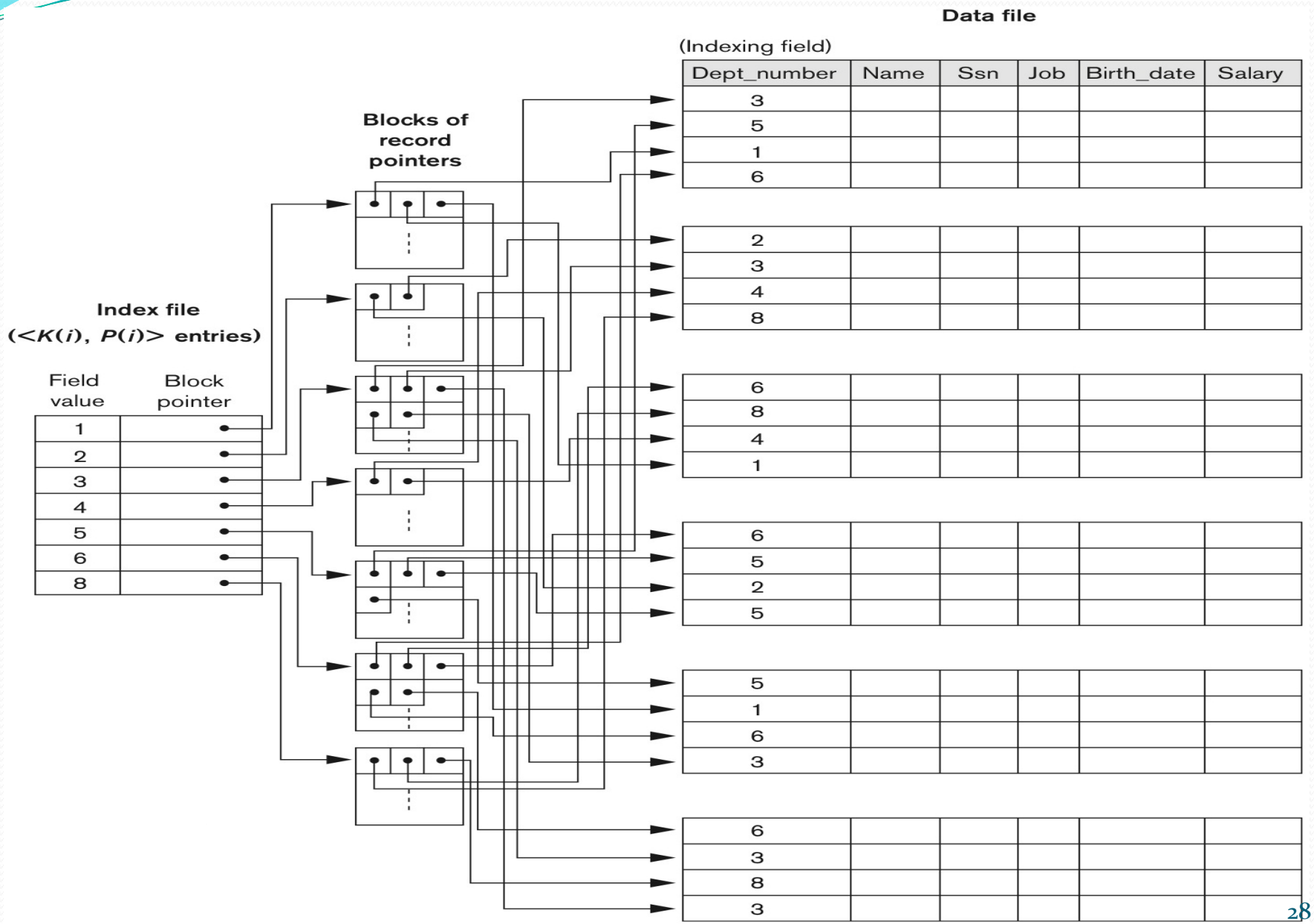
- A **secondary index** provides a secondary means of accessing a data file for which some primary access already exists.
- The secondary index may be created on a field that is a candidate key and has a unique value in every record, or on a nonkey field with duplicate values.
- It is an Ordered file with two fields
  - Indexing field, *K(i)*
  - Block pointer or record pointer, *P(i)*
- *Many* secondary indexes (and hence, indexing fields) can be created for the same file—each represents an additional means of accessing that file based on some specific field.

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

Secondary Indexes (on a key field)



Secondary Indexes (on a non-key field)



	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

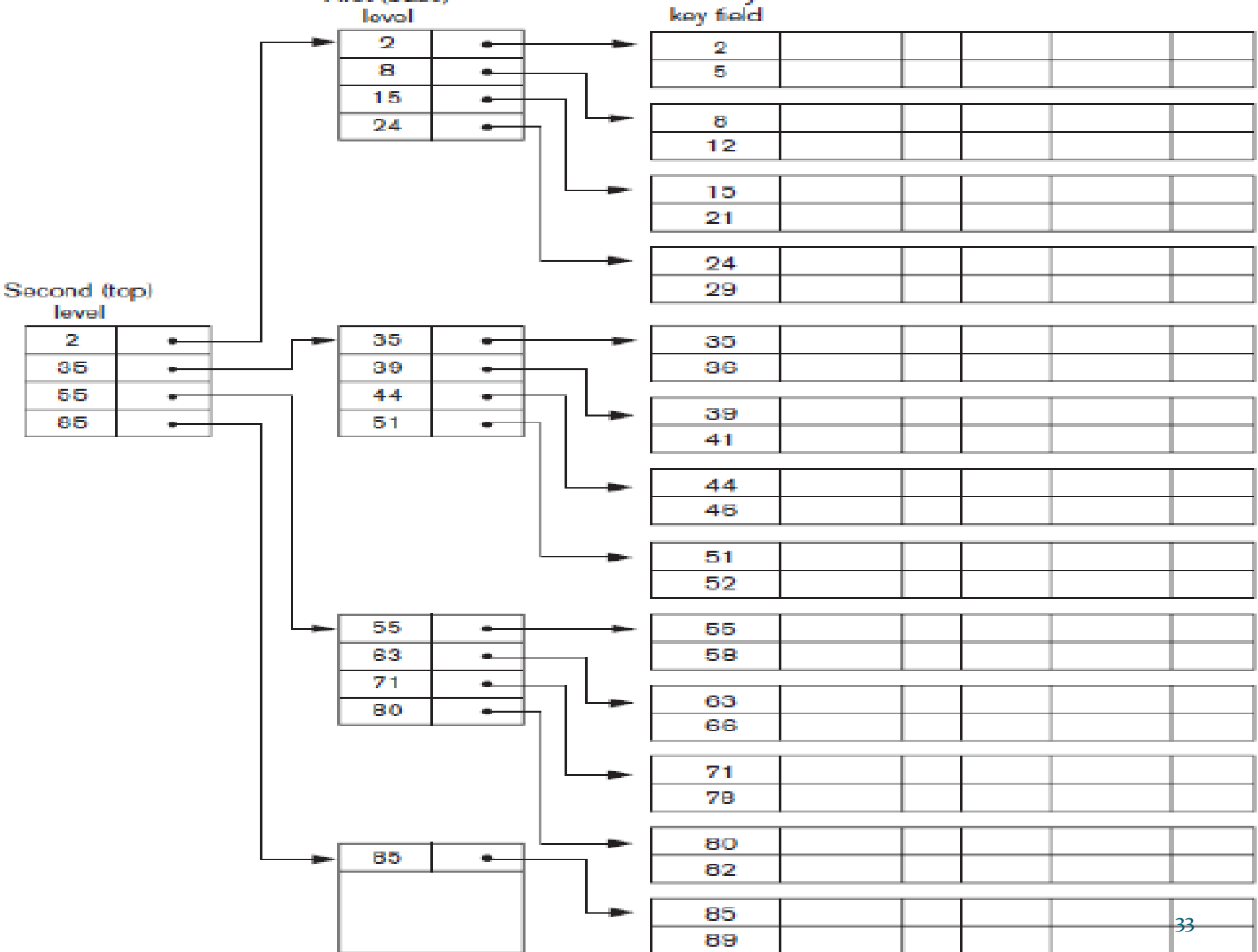
Multilevel Indexes

- Because a single-level index is an ordered file, we can create a primary index to the index itself
  - the original index file is called the first-level index
  - the index to the index is called the second-level index

We can repeat the process, creating a 3rd, 4th,…. top level until all entries in the top level fit in one disk block

- A multi-level index can be created for any type of first level index (primary, secondary, clustering) as long as the first-level index consists of more than one disk blocks Such a multi-level index is a form of search tree.
- however, insertion and deletion of new index entries is a severe problem because every level of the index is an ordered file Hence most multi-level indexes use B-tree or B+ tree data structures, which leave space in each tree node disk block) to allow for new index entries

- A multi-level index considers the index file as an ordered file with a distinct entry for each K(i)
- First level We can create a primary index for the first level
  - Second level
  - Because the second level uses block anchors we only need an entry for each block in the first level We can repeat this process for the second level
- Third level would be a primary level for the second level And so on … until all the entries of a level fit in a single block

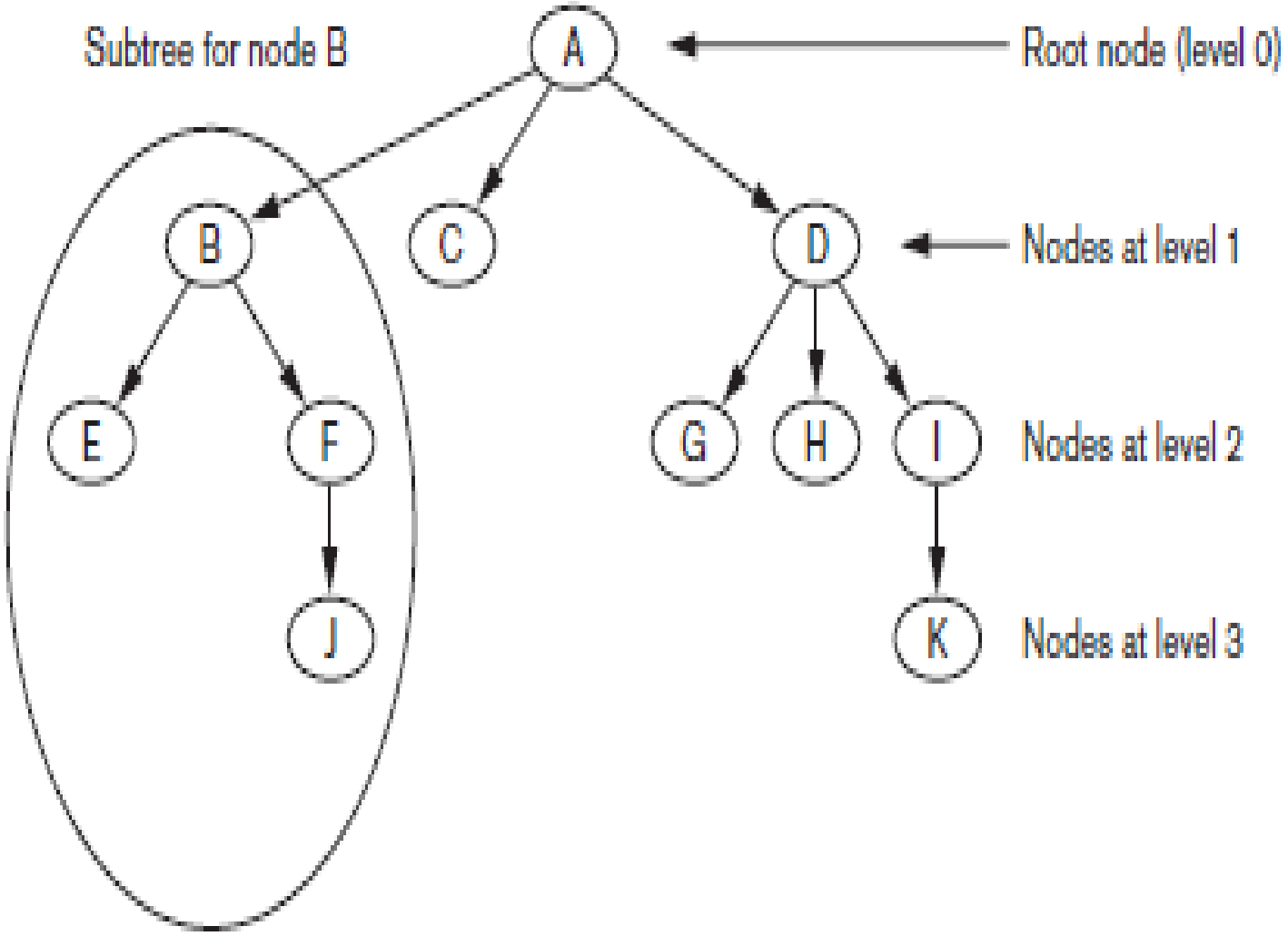

level
key field
2
8
15
24
2
5
8
12
15
21
24
29
Second (top)
level
2
35
55
85
35
39
44
51
35
36
39
41
44
46
51
52
55
63
71
80
55
58
63
66
71
78
80
82
85
89

- Such a multi-level index is a form of search tree
  - However, insertion and deletion of new index entries is a severe problem because every level of the index is an ordered file

Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- B-trees and B+-trees are special cases of the well-known search data structure known as a **tree**.
- A tree is formed of nodes. Each node in the tree, except for a special node called the root, has one parent node and zero or more child nodes. The root node has no parent. A node that does not have any child nodes is called a leaf node; a nonleaf node is called an internal node.
- The level of a node is always one more than the level of its parent, with the level of the root node being zero. A subtree of a node consists of that node and all its descendant nodes—its child nodes, the child nodes of its child nodes, and so on.

A tree data structure that shows an unbalanced tree.

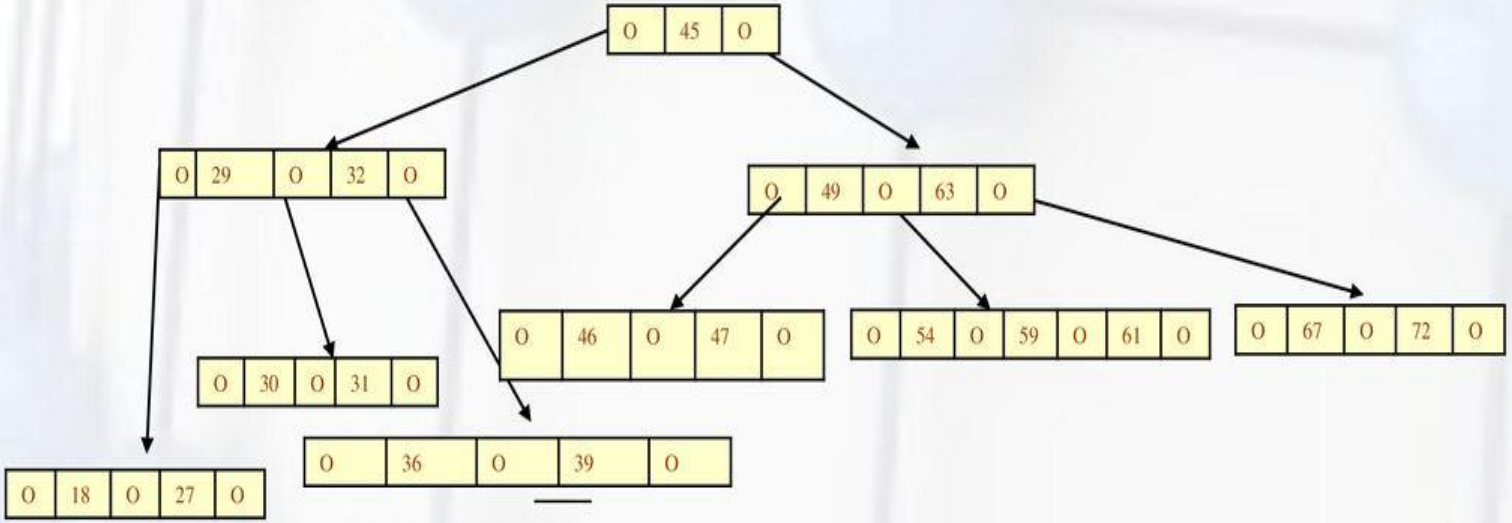


(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

B-Trees

- Provide multi-level access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
  - Each node is at least half-full
- Each node in a B-tree of order p can have at most p-1 search values

Example of B-Tree with m = 4



Insertion operation on B-Trees

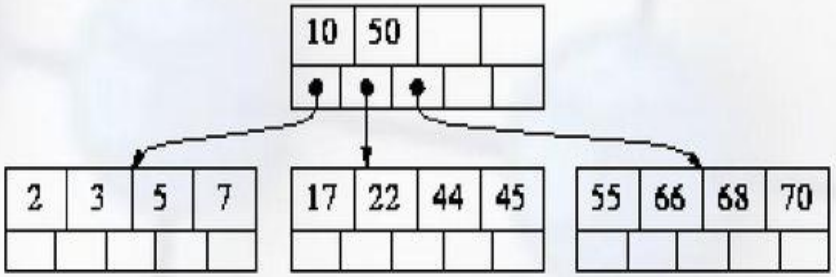
Algorithm

Step 1. Search the B tree to find the leaf node where the new key value should be inserted.

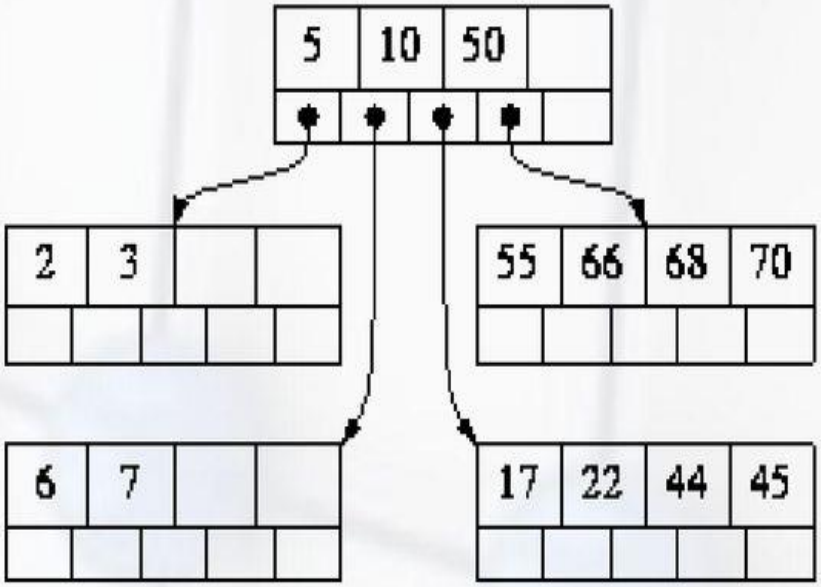
Step 2. If the node is not full, that is it contains less than m-1 key values then insert the new element in the node, keeping the node's elements ordered.

Step 3. If the node is full, insert the new value in order into the existing set of keys, split the node at its median into two nodes. note that the split nodes are half full. Insert the median element to its parent's node. Go to step 2.

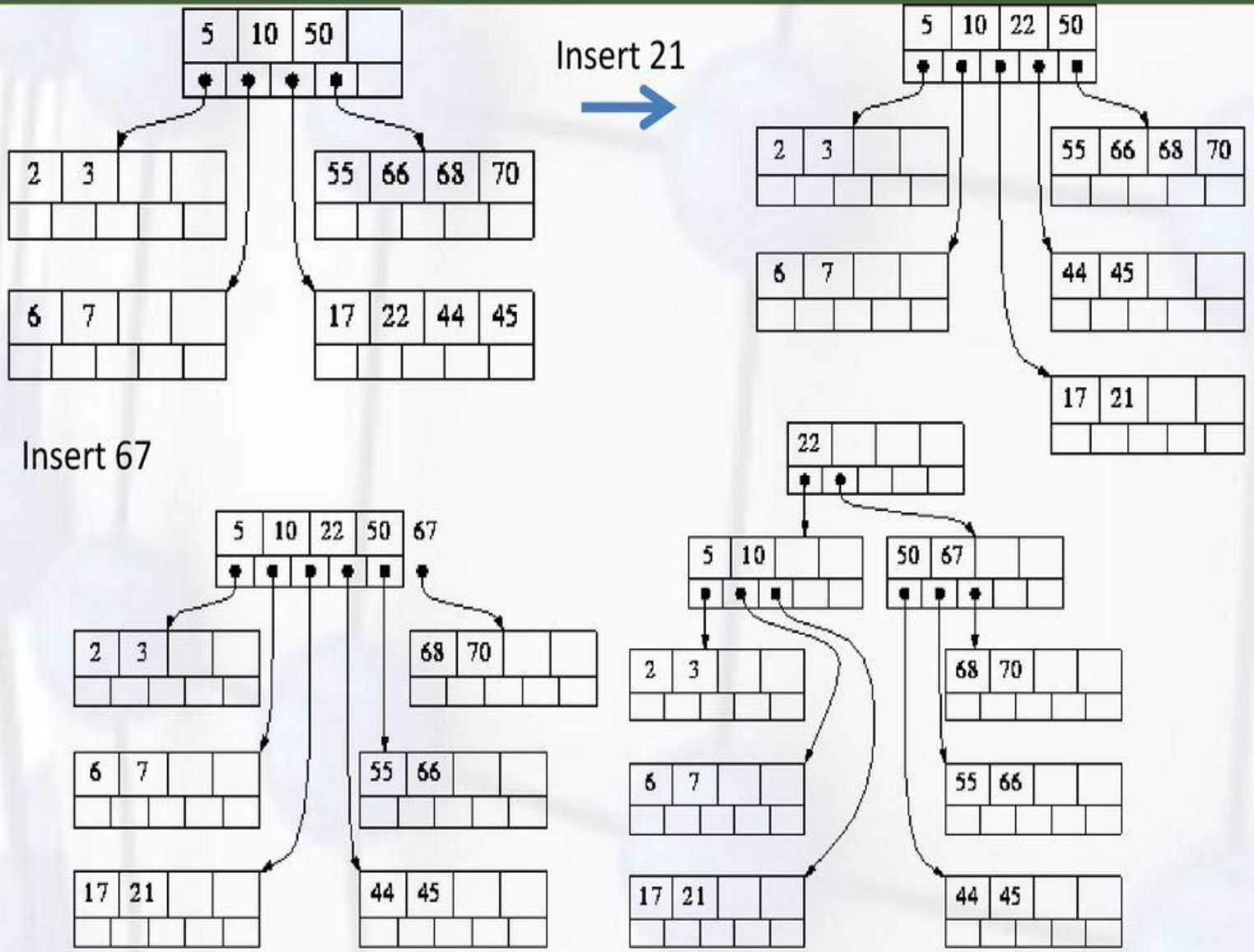
Example of B-Tree insertion



Insert 6



Example of B-Tree insertion



B tree deletion

Before going through the steps below, a B tree of degree **m**.

1. A node can have a maximum of m children. (i.e. 3)

2. A node can contain a maximum of $m - 1$ keys. (i.e. 2)

3. A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)

4. A node (except root node) should contain a minimum of $\lceil m/2 \rceil - 1$ keys. (i.e. 1)

B-tree Deletion

Algorithm

Step 1. Locate the leaf node which has to be deleted

Step 2. If the leaf node contains more than minimum number of key values (more than m/2 elements), then delete the value.

Step 3. else if the leaf node does not contain even m/2 elements then, fill the node by taking an element either from the left or from the right sibling

Step 3.1 If the left sibling has more than the minimum number of key values (elements), push its largest key into its parent's node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

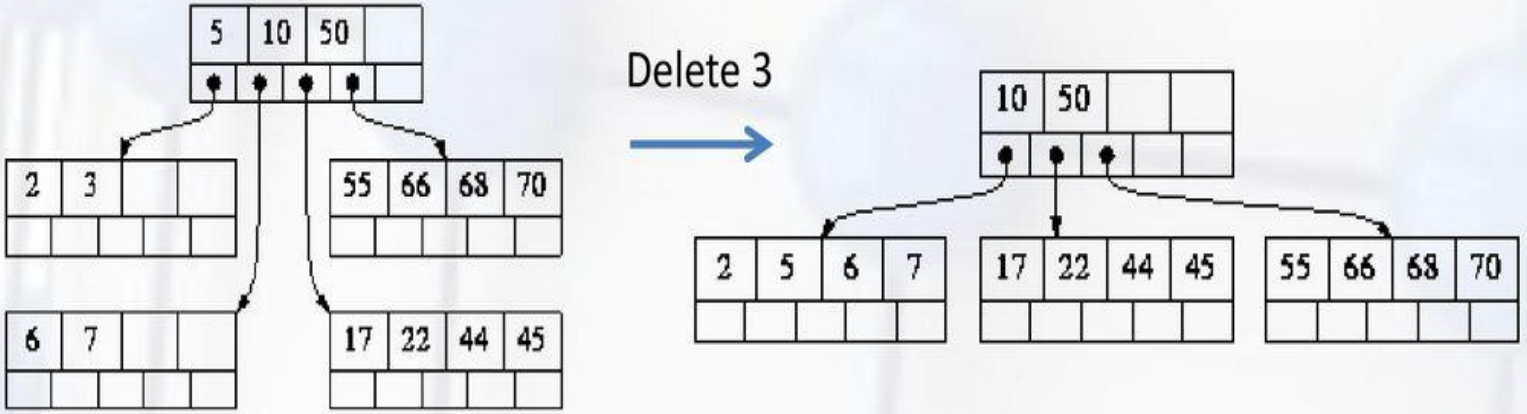
Step 3.2 else if the right sibling has more than the minimum number of key values (elements), push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

B-tree Deletion

Step 4. else if both left and right siblings contain only minimum number of elements, then create a new leaf node by combining the two leaf nodes and the intervening element of the parent node (ensuring that the number of elements do not exceed the maximum number of elements a node can have, that is, m). If pulling the intervening element from the parent node leaves it with less than minimum number of keys in the node, then propagate the process upwards thereby reducing the height of the B tree.

- To delete an internal node, promote the successor or predecessor of the key to be deleted to occupy the position of the deleted key. This predecessor or successor will always be in the leaf node. So further the processing will be done as if a value from the leaf node has been deleted.

Example of B-tree Deletion



B+-Trees

- A **B+ Tree** is primarily utilized for implementing dynamic indexing on multiple levels.
- Compared to B- Tree, the B+ Tree stores the data pointers only at the leaf nodes of the Tree, which makes search more process more accurate and faster.

B+ Tree vs. B Tree

B+ Tree	B Tree
Search keys can be repeated.	Search keys cannot be redundant.
Data is only saved on the leaf nodes.	Both leaf nodes and internal nodes can store data
Data stored on the leaf node makes the search more accurate and faster.	Searching is slow due to data stored on Leaf and internal nodes.
Deletion is not difficult as an element is only removed from a leaf node.	Deletion of elements is a complicated and time-consuming process.
Linked leaf nodes make the search efficient and quick.	You cannot link leaf nodes.

Rules for B+ Tree

- Here are essential rules for B+ Tree.
- Leaves are used to store data records.
- It stored in the internal nodes of the Tree.
- If a target key value is less than the internal node, then the point just to its left side is followed.
- If a target key value is greater than or equal to the internal node, then the point just to its right side is followed.
- The root has a minimum of two children.

New Database Applications and Architectures

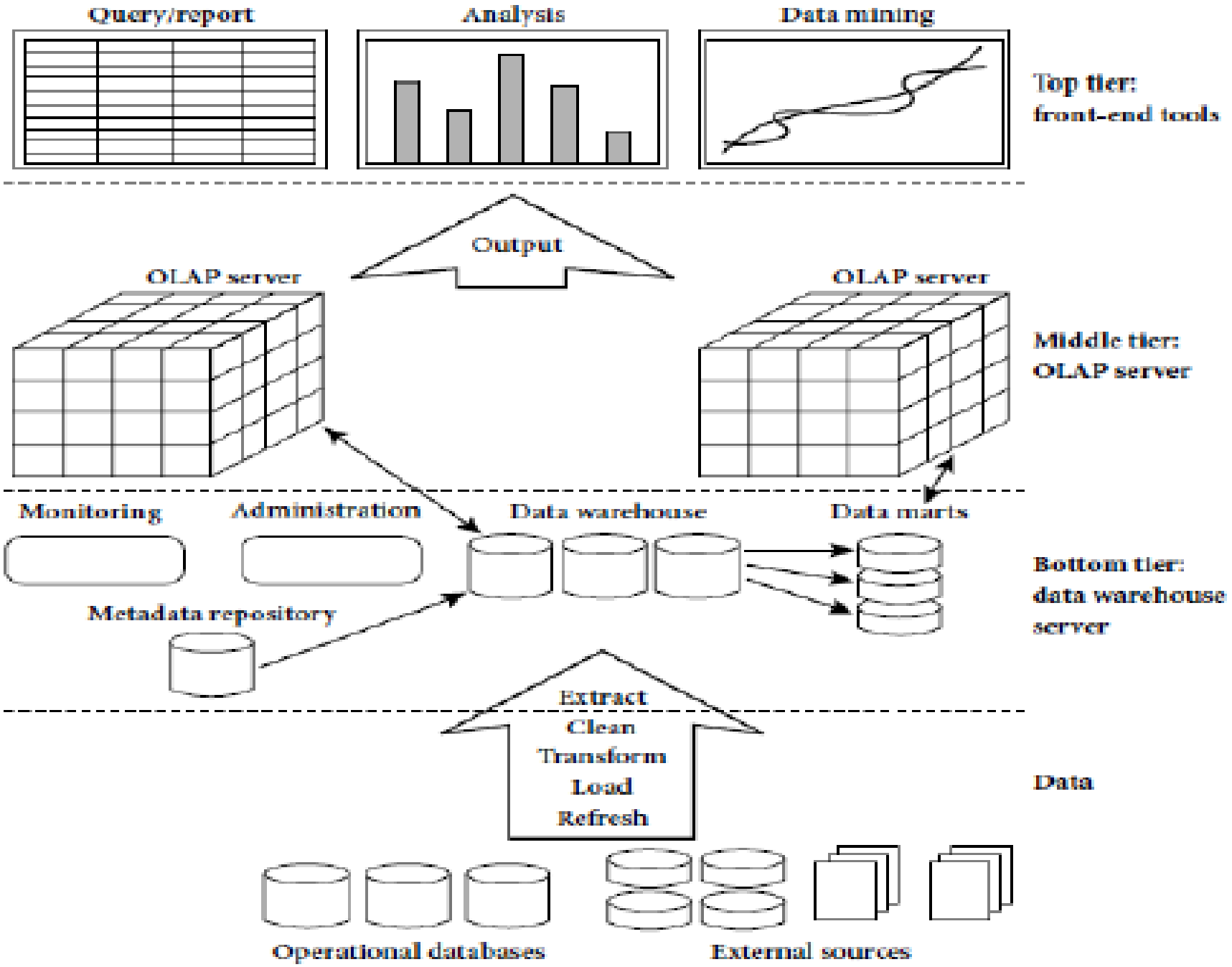
Data Warehousing

- We defined a database as a collection of related data and a database system as a database and database software together. A data warehouse is also a collection of information as well as a supporting system.

- A **data warehouse** as a subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions.

Data Warehousing

- **Subject Oriented**: Data that gives information about a particular subject instead of about a company's ongoing operations.
- **Integrated:** Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.
- **Time-variant:** All data in the data warehouse is identified with a particular time period.
- **Non-volatile:** Data is stable in a data warehouse. More data is added but data is never removed.


Query/report
Analysis
Data mining
Top tier:
front-end tools
Output
OLAP server
OLAP server
Middle tier:
OLAP server
Monitoring
Administration
Data warehouse
Data marts
Bottom tier:
data warehouse
server
Metadata repository
Extract
Clean
Transform
Load
Refresh
Data
Operational databases
External sources

- **Tier-1:**
- The bottom tier is a warehouse database server that is almost always a relationaldatabase system. Back-end tools and utilities are used to feed data into the bottomtier from operational databases or other external sources (such as customer profile information provided by external consultants). These tools and utilities perform data extraction, cleaning, and transformation (e.g., to merge similar data from different sources into a unified format), as well as load and refresh functions to update the data warehouse . The data are extracted using application program interfaces known as gateways. supported by the underlying DBMS and allows client programs to generate SQL code to be executed at a server.

- Examplesof gateways include ODBC (Open Database Connection) and OLEDB (Open Linkingand Embedding for Databases) by Microsoft and JDBC (Java Database Connection). This tier also contains a metadata repository, which stores information aboutthe data warehouse and its contents.

- Tier-2: The middle tier is an OLAP server that is typically implemented using either a relational OLAP (ROLAP) model or a multidimensional OLAP. OLAP model is an extended relational DBMS that maps operations on multidimensional data to standard relational operations. A multidimensional OLAP (MOLAP) model, that is, a special-purpose server that directly implements multidimensional data and operations.
- Tier-3: The top tier is a front-end client layer, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).

Characteristics of data warehouses

- Multidimensional conceptual view
- Unlimited dimensions and aggregation levels
- Unrestricted cross-dimensional operations
- Dynamic sparse matrix handling
- Client/server architecture
- Multiuser support
- Accessibility
- Transparency
- Intuitive data manipulation
- Inductive and deductive analysis
- Flexible distributed reporting

Data Warehouse Models

1. Enterprise warehouse:

- An enterprise warehouse collects all of the information about subjects spanning the entire organization.
- It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope.
- It typically contains detailed data as well as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond.
- An enterprise data warehouse may be implemented on traditional mainframes, computer super
- servers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

2. Data mart:

- A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized.

- Data marts are usually implemented on low-cost departmental servers that are UNIX/LINUX- or Windows-based. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years. However, it may involve complex integration in the long run if its design and planning were not enterprise-wide.

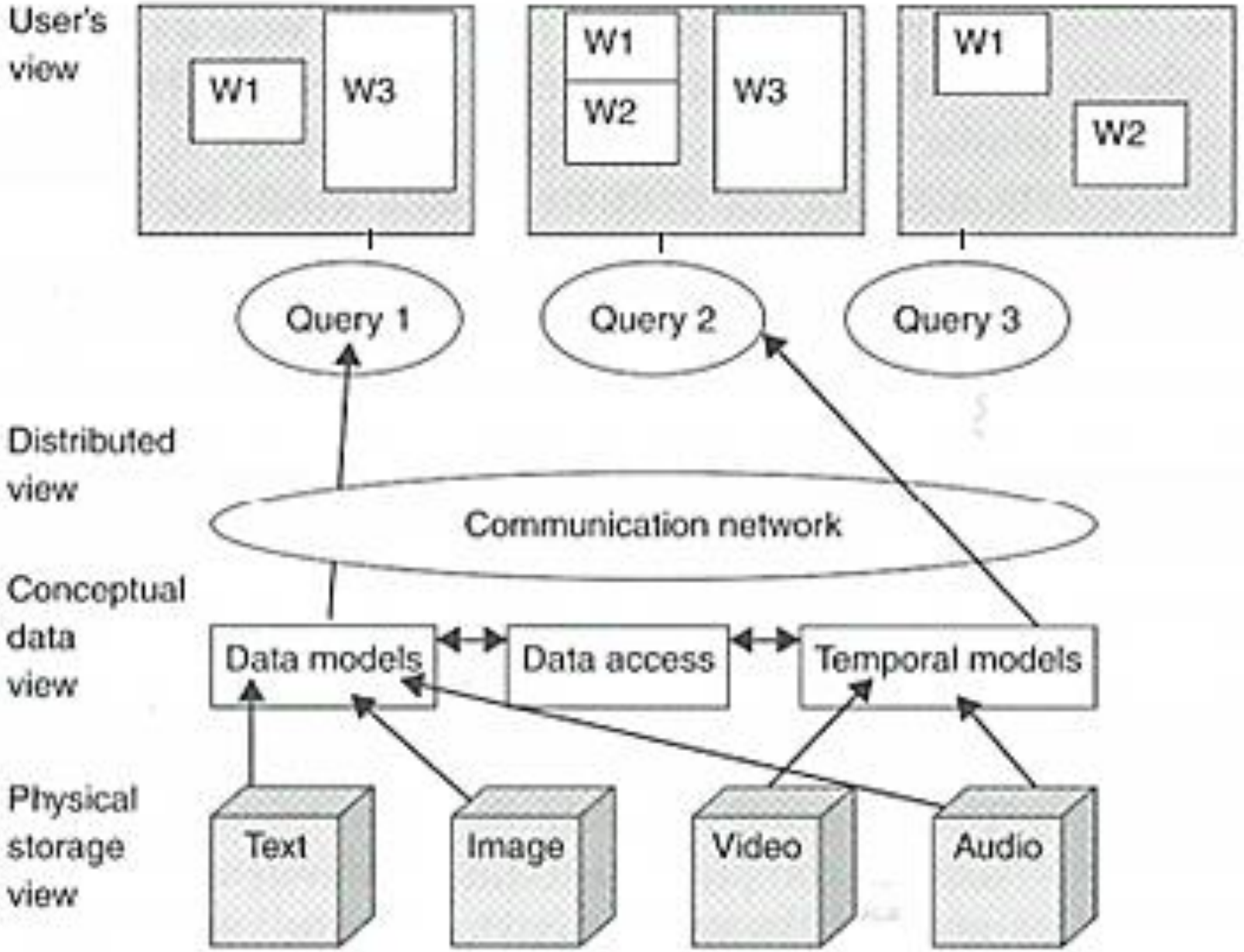
3. Virtual warehouse:

- A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but requires excess capacity on operational database servers.

Multimedia Database Concepts

- Multimedia database is a collection of multimedia data which includes text, images, graphics (drawings, sketches), animations, audio, video, among others. These databases have extensive amounts of data which can be multimedia and multisource.

- The framework which manages these multimedia databases and their different types so that the data can be stored, utilized, and delivered in more than one way is known as a multimedia database management system.



Multimedia databases – user, conceptual and physical storage views

- *Physical Storage View*:
  - how multimedia objects are stored in a file system.
  - Since multimedia objects are typically huge, different techniques needed for their storage as well as retrieval.
- *Conceptual Data View*:
  - Describes the interpretations created from physical storage representation of media objects.
    - Needed because most object are just *Binary Large Objects* (BLOBs).
  - Also deals with the issue of providing fast access to stored data by means of index mechanisms.
- *Distributed View*:
  - MM objects might be stored in different systems.
  - Systems and users might access stored data over computer networks.

- *User's View*:
  - Objects retrieved from the database(s) have to be appropriately presented.
  - Though these views are true for a traditional database management system, diverse characteristics of media objects introduce many interesting issues.

Multimedia Database Applications

- **Documents and record management:** Industries which keep a lot of documentation and records. Ex: Insurance claim industry.
- **Knowledge dissemination:** Multimedia database is an extremely efficient tool for knowledge dissemination and providing several resources. Ex: electronic books
- **Education and training:** Multimedia sources can be used to create resources useful in education and training. These are popular sources of learning in recent days. Ex: Digital libraries.
- **Real-time monitoring and control:** Multimedia presentation when coupled with active database technology can be an effective means for controlling and monitoring complex tasks. Ex: Manufacture control

Mobile Database

- Mobile Database is a database that is transportable, portable, and physically separate or detached from the corporate database server but has the capability to communicate with those servers from remote sites allowing the sharing of various kinds of data.
- With mobile databases, users have access to corporate data on their laptop, PDA, or other Internet access device that is required for applications at remote sites.

Features of Mobile database

The features of the mobile database as follows:

- A cache is maintained to hold frequent and transactions so that they are not lost due to connection failure.
- As the use of laptops, mobile and PDAs is increasing to reside in the mobile system.
- Mobile databases are physically separate from the central database server.
- Mobile databases resided on mobile devices.

- Mobile databases are capable of communicating with a central database server or other mobile clients from remote sites.
- With the help of a mobile database, mobile users must be able to work without a wireless connection due to poor or even non-existent connections (disconnected).
- A mobile database is used to analyze and manipulate data on mobile devices.

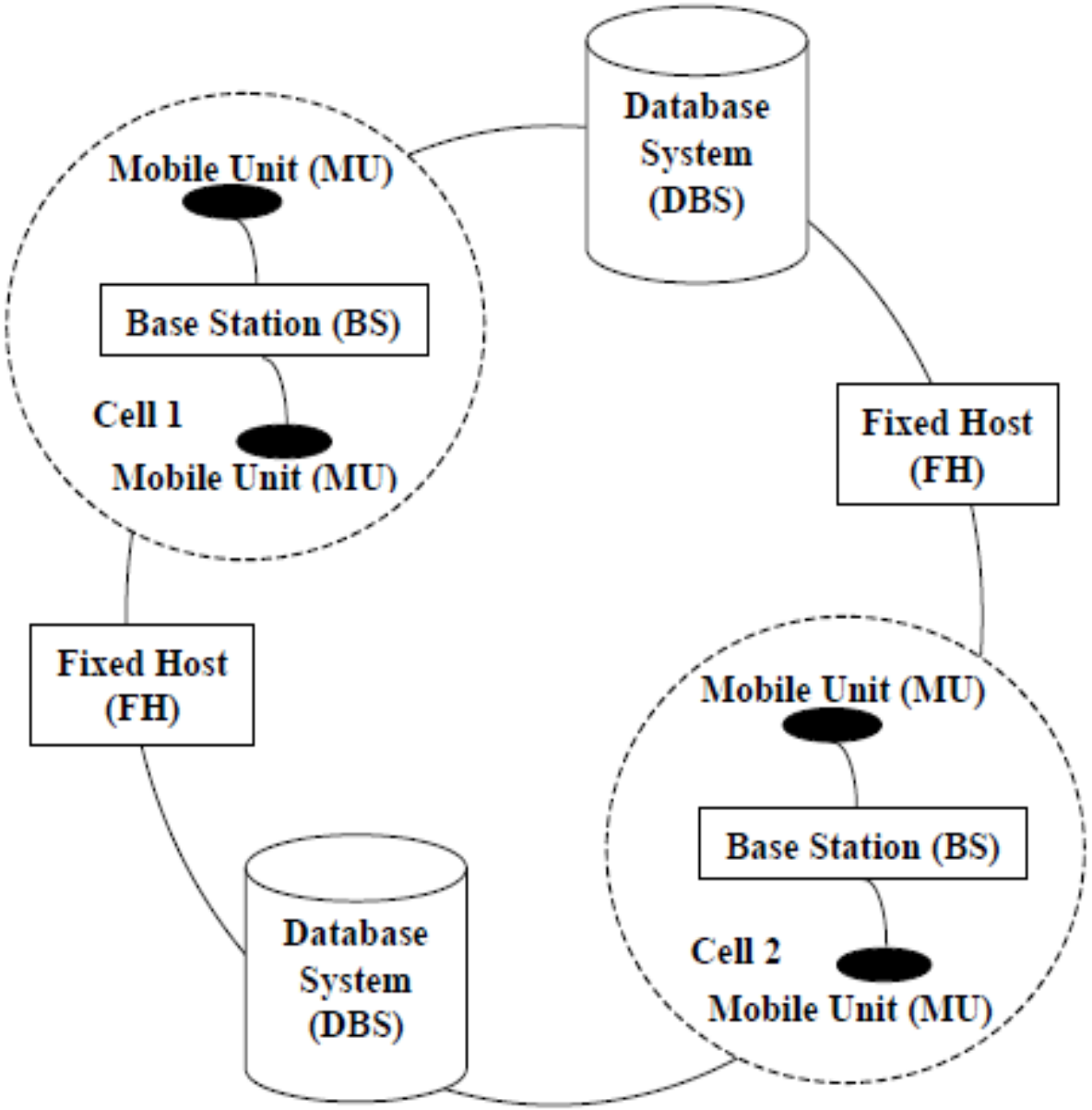


Fig: Mobile System Architecture

- The Mobile Database System Architecture consists of Database Systems (DBs), Fixed Host (FHs) and No. of cells.
- Each cell is locally connected with a Base Station (BS) and Mobile Units (MUs). Those cells are globally connected with Database Systems and Fixed Host.

- **Fixed Hosts –**
  It performs the transactions and data management functions with the help of database servers.

- **Mobiles Units –**
  These are portable computers that move around a geographical region that includes the cellular network that these units use to communicate to base stations.

- **Base Stations –**
  These are two-way radios installation in fixed locations, that pass communication with the mobile units to and from the fixed hosts.

NoSQL

- NOSQL n Not only SQL n Most NOSQL systems are distributed databases or distributed storage systems
- Focus on semi-structured data storage, high performance, availability, data replication, and scalability.
- NOSQL systems focus on storage of “big data”
- Typical applications that use NOSQL
  - Social media
  - Web links
  - User profiles
  - Marketing and sales
  - Posts and tweets
  - Road maps and spatial data
  - Email

Property

- Traditional DBMS
  - ACID
- NoSQL
  - BASE
    - Basically Available
    - Soft State
    - Eventual Consistency

- BigTable n Google's proprietary NOSQL system
- Column-based or wide column store
- DynamoDB (Amazon)
- Key-value data store
- Cassandra (Facebook)
- Uses concepts from both key-value store and column-based systems

- MongoDB and CouchDB
- Document stores
- Neo4J and GraphBase
- Graph-based NOSQL systems
- OrientDB
- Combines several concepts
- Database systems classified on the object model
- Or native XML model

NOSQL characteristic

- Scalability
- Availability,
- replication, and eventual consistency
- Replication models
- Master-slave
- Master-master
- Sharing of files
- High performance data access

- NOSQL characteristics related to data models and query languages n
  - Schema not required
  - Less powerful query languages
  - Versioning

Categories of NOSQL systems

- Document-based NOSQL systems
  - Mongodb
    - Uber
- NOSQL key-value stores
  - Redis
    - Twitter
- Column-based or wide column NOSQL systems
  - Cassandra
    - Netflix
- Graph-based NOSQL systems
  - Neo4J
    - Walmart
- Hybrid NOSQL systems
- Object databases
- XML databases

Native XML database

- It's a database that stores and retrieves XML documents efficiently
- The view that users have should be that it stores and retrieves XML documents
- The query language should be based on XML, and the structure of XML
- The indexes should ensure that the fast execution of queries against XML documents

What are the implications of the definition?

- A native XML database is specialized for storing XML documents and stores all components of the XML model intact.
- XML documents go in and XML documents come out.
- The underlying data storage format or the physical model is unimportant for the categorization of databases

Advantages of storing data in NXD

- Semistructured data stored in a relational database will result in a large number of nulls or a large number of tables.
- Retrieval of documents or parts of documents might be fast.
- Retrieving a view over the data might be slower

Architectures for NXDs

- Architectures for NXDs fall into two categories
  - Text based NXD
  - Model based NXD

Text based NXD

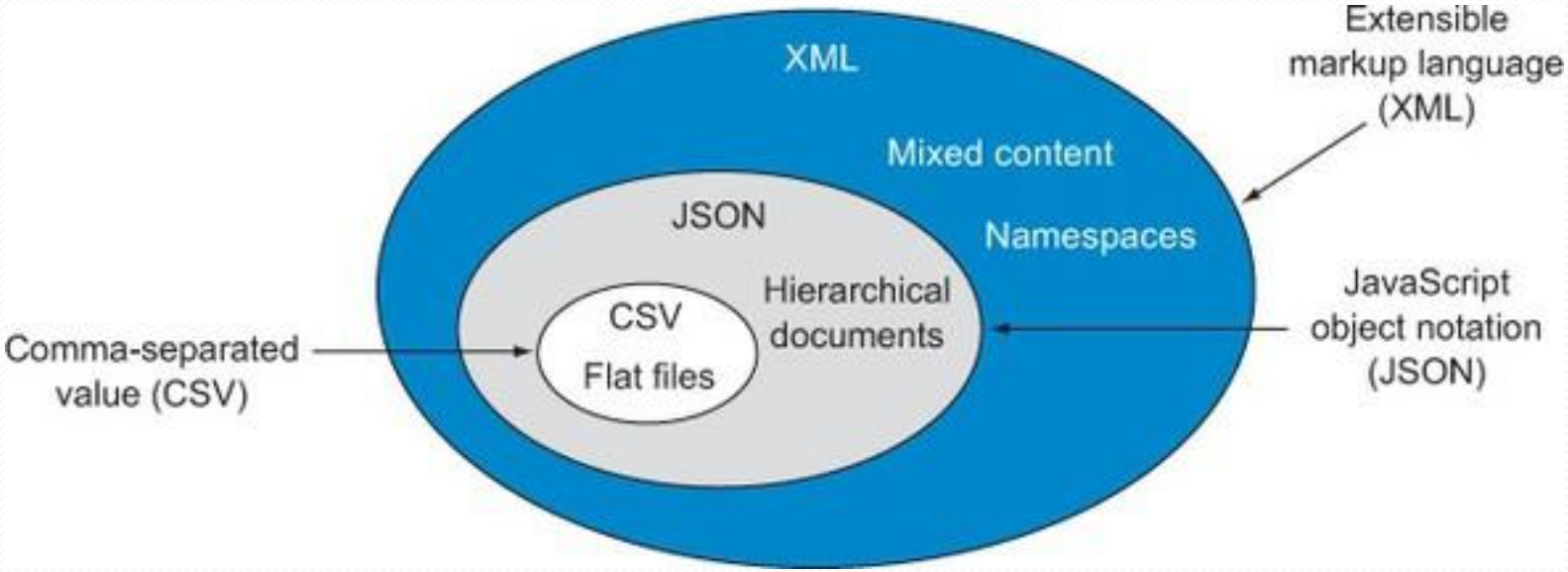
- Stores XML as text e.g. in a file, in a relational database, in some other form.
- Usually have indexes, allowing direct access within the XML document, improving access to documents or pieces of documents.
- Problem when inverting the hierarchy or portions of it.

Model based NXD

- Rather than storing the XML document as text, build an internal object model from the document and store this model.
- How the model is stored depends on the underlying database. Some databases use a propriety storage format optimized for their model.
- Tables can include doc, node, element_name, element_value, etc.

- As mentioned, XML files are the best choice when your document includes mixed content.
- XML also supports an often-controversial feature: namespaces. Namespaces allow you to mix data elements from different domains within the same document, yet retain the source meaning of each element.
- Documents that support multiple namespaces allow applications to add new elements in new namespaces without disrupting existing data queries.

Document Orientated Databases



- **The expressiveness of three document formats.**
  - **Comma-separated value (CSV) files are designed to only store flat files that don't contain hierarchy.**
  - **JavaScript Object Notation (JSON) files can store flat files as well as hierarchical documents.**
  - **Extensible Markup Language (XML) files can store flat files, hierarchical documents, and documents that contain mixed content and namespaces.**

Graph Database

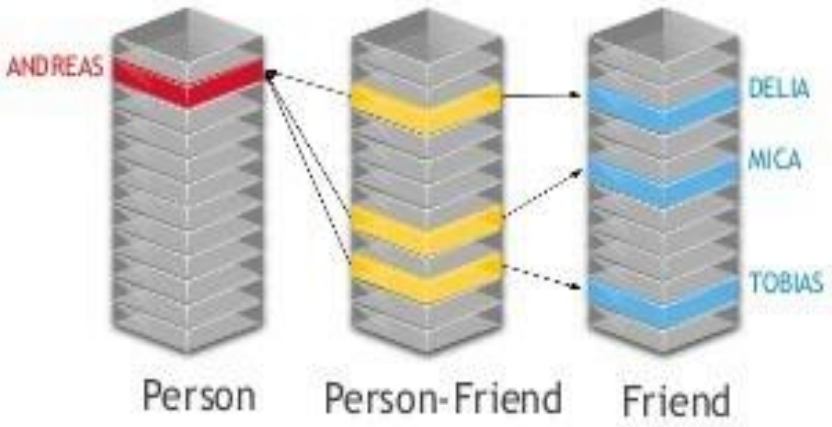
- A graph database stores nodes and relationships instead of tables, or documents. Data is stored just like you might sketch ideas on a whiteboard.
- Your data is stored without restricting it to a pre-defined model, allowing a very flexible way of thinking about and using it.
- The idea stems from graph theory in mathematics, where graphs represent data sets using **nodes**, **edges**, and **properties**.

Graph Database

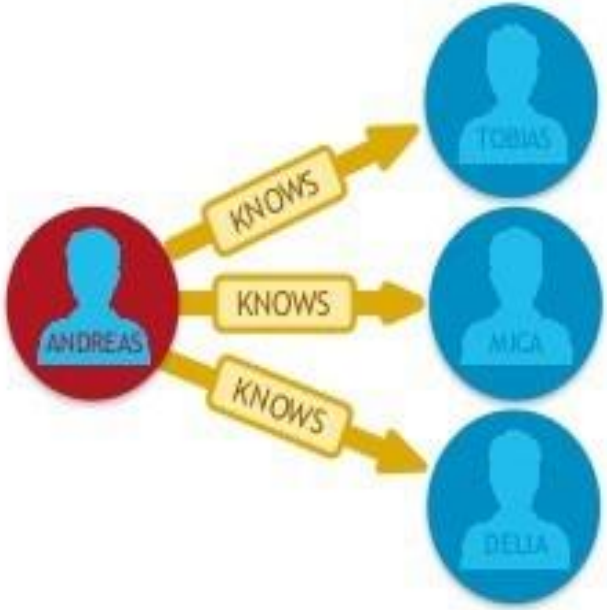
- **Nodes** or points are instances or entities of data which represent any object to be tracked, such as people, accounts, locations, etc.
- **Edges** or lines are the critical concepts in graph databases which represent relationships between nodes. The connections have a direction that is either unidirectional (one way) or bidirectional (two way).
- **Properties** represent descriptive information associated with nodes. In some cases, edges have properties as well.

Relational Versus Graph Models

Relational Model



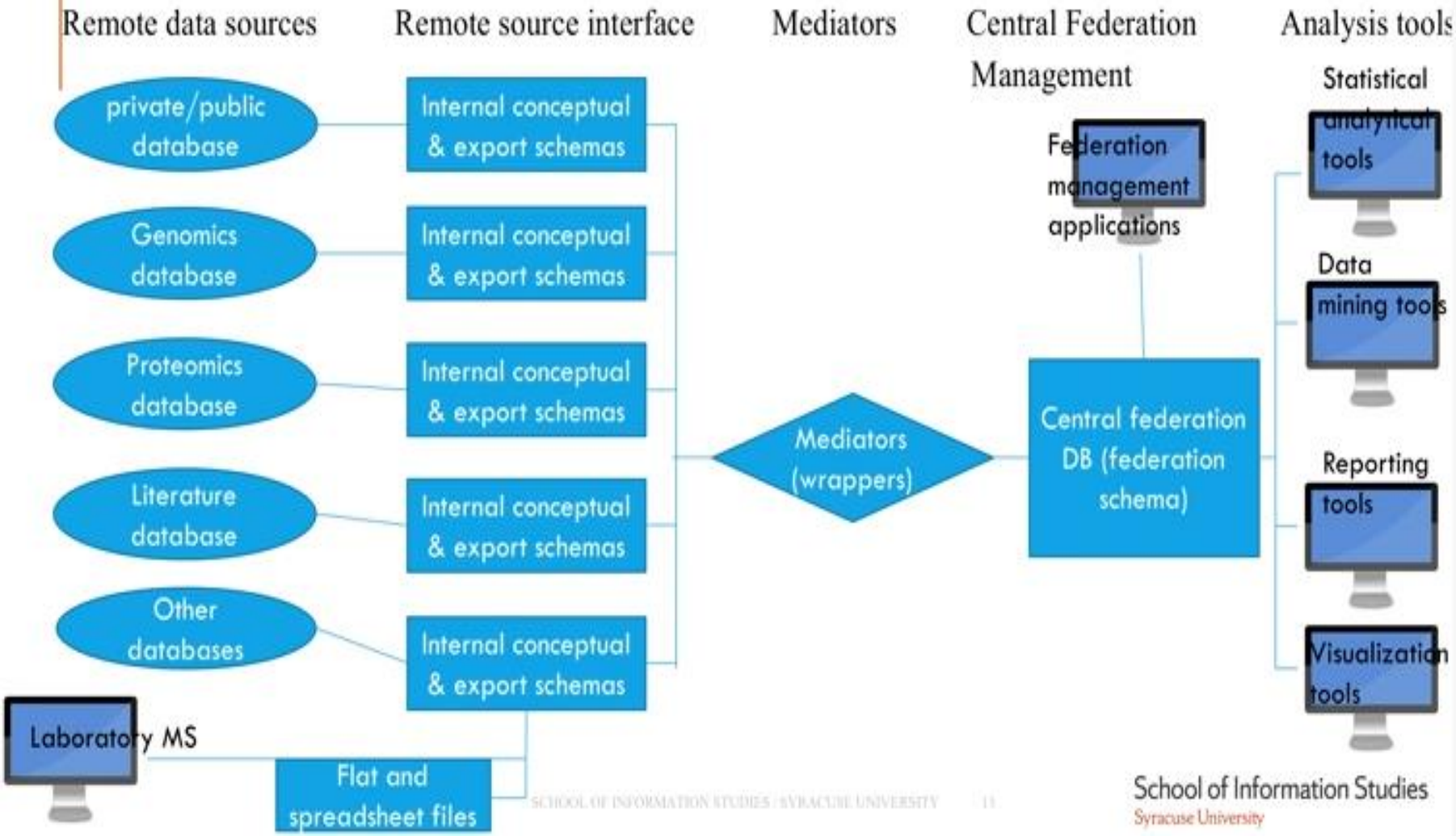
Graph Model



Federated database system

- A **federated database system** is a type of meta-database management system (DBMS), which transparently integrates multiple autonomous database systems into a single **federated database**.
- The constituent databases are interconnected via a computer network and may be geographically decentralized. Since the constituent database systems remain autonomous, a federated database system is a contrastable alternative to the (sometimes daunting) task of merging together several disparate databases.
- A federated database, or **virtual database**, is the fully integrated, logical composite of all constituent databases in a federated database system.

Federated database architecture



Five Level Schema Architecture for FDBSs

The five level schema architecture includes the following:

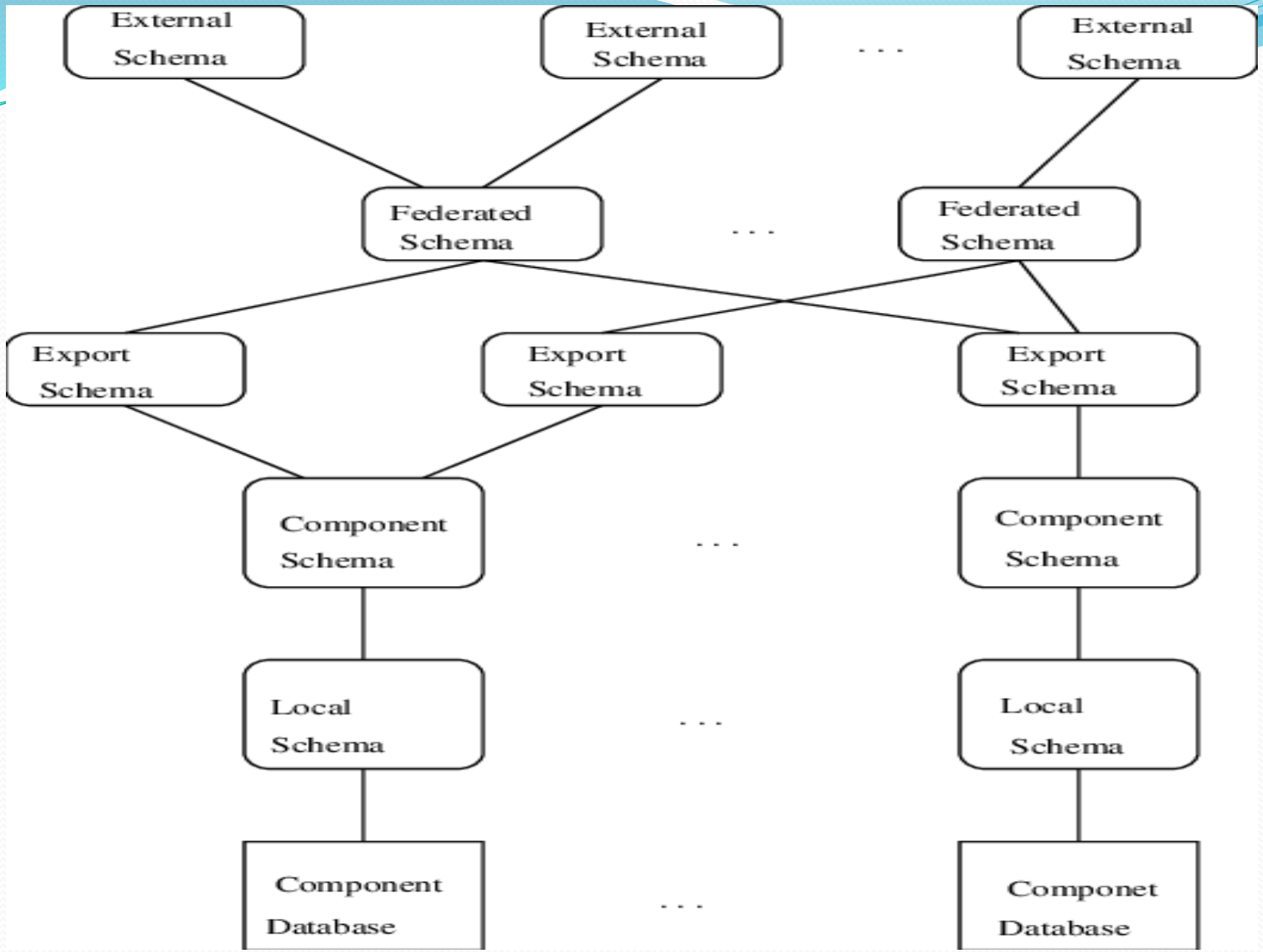
- Local Schema is the conceptual concept expressed in primary data model of component DBMS.

- Component Schema is derived by translating local schema into a model called the canonical data model or common data model. They are useful when semantics missed in local schema are incorporated in the component. They help in integration of data for tightly coupled FDBS.

- Export Schema represents a subset of a component schema that is available to the FDBS. It may include access control information regarding its use by specific federation user. The export schema help in managing flow of control of data.

• Federated Schema is an integration of multiple export schema. It includes information on data distribution that is generated when integrating export schemas.

•External Schema defines a schema for a user/applications or a class of users/applications



Five Level Schema Architecture for FDBSs

- Federated architectures differ based on levels of integration with the component database systems and the extent of services offered by the federation. A FDBS can be categorized as loosely or tightly coupled systems.

  - Loosely Coupled require component databases to construct their own federated schema. A user will typically access other component database systems by using a multidatabase language but this removes any levels of location transparency, forcing the user to have direct knowledge of the federated schema. A user imports the data they require from other component databases and integrates it with their own to form a federated schema.

- Tightly coupled system consists of component systems that use independent processes to construct and publicize an integrated federated schema