Record Storage, File Organization, and Indexes

ISM6217 - Advanced Database

Physical Database Design Phase

- Inputs into the Physical Design Phase
  - Logical (implementation) model
  - Documentation/Definitions
  - DBMS Characteristics
  - Response time requirements
  - Security, Backup, Recovery, Retention, Integrity requirements
- Logical Design Phase focuses on *What*; Physical Design Phases focuses on *How*.

Physical Database Design Phase

- Outputs:
  - Produces a Description of the Implementation of the Database on Secondary Storage.
  - Describes the storage structures and access methods used to achieve efficient access to the data.
- Focus on Data Processing Efficiency

Steps in Physical Database Design

- Translate Logical (Implementation) Data Model for Target DBMS
  - Design base relations and constraints

Steps in Physical Database Design

- Design Physical Representation
  - Analyze transactions
  - Choose file organizations
  - Choose secondary indexes
  - Introduction of controlled redundancy (Denormalization)
  - Estimate disk space requirements

Our focus today

- There are additional steps that we'll talk about next week.

Files and Records

- Records
  - Contain fields (fixed and variable length fields)
  - Fixed-length
  - Variable-length
  - Spanned
  - Unspanned

Files and Records

- Block (page): The amount of data read or written in one I/O operation.
- Blocking Factor: The number of physical records per block.
- File Organization: How the physical records in a file are arranged on the disk.
- Access Method: How the data can be retrieved based on the file organization.

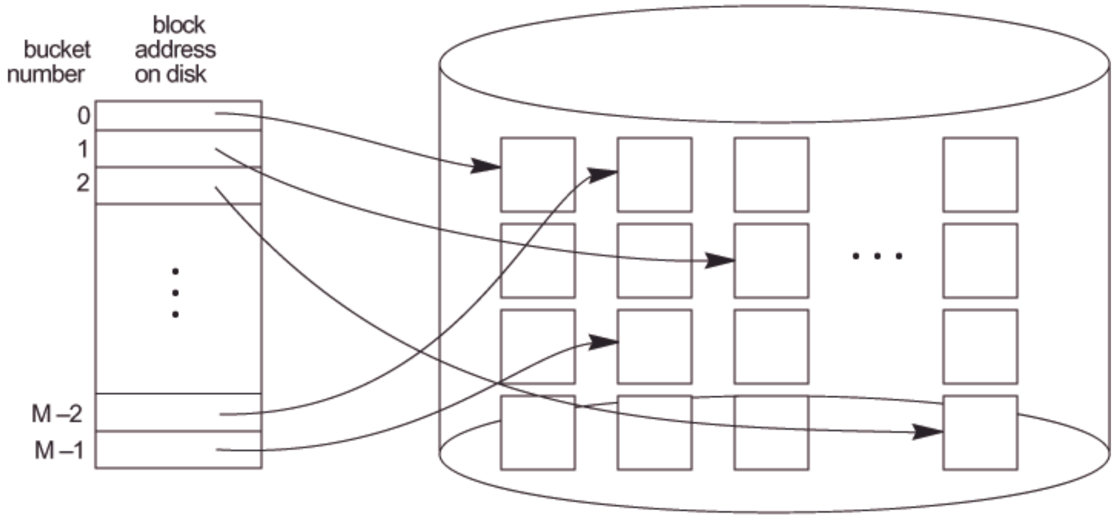
Index Classification

- Primary vs. secondary: If search key contains primary key, then called primary index.
  - Unique index: Search key contains a candidate key.
- Clustered vs. unclustered: If order of data records is the same as, or `close to', order of data entries, then called clustered index.
  - A file can be clustered on at most one search key.
  - Cost of retrieving data records through index varies greatly based on whether index is clustered or not!

Hash-Based Indexes

- Hash function calculates the address of the page on which the record is stored.
- The field that's used in the hash function is called the *hash field*.
  - Called a *hash key* if the hash field is also a key field.
- Good for equality searches
- Not good for range searches

Hash File



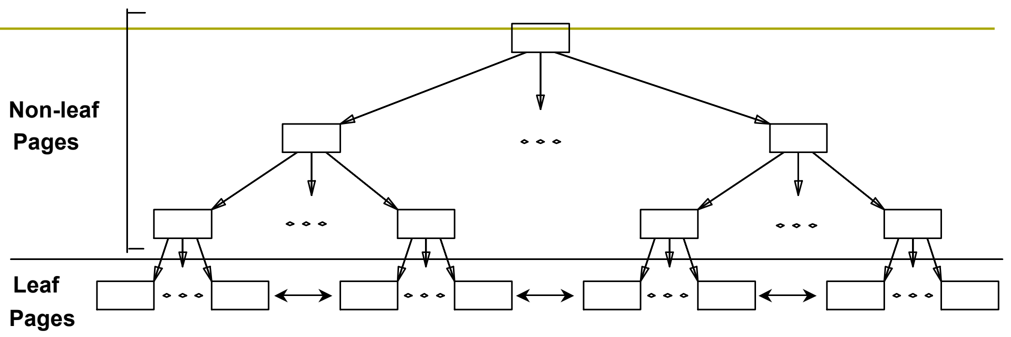
Tree Indexes

- Many DBMS use tree data structures to hold data/indexes.
- Tree contains a hierarchy of nodes
- Root node has no parent
- Other nodes have one parent
- Nodes can (but don’t have to) have child nodes.
- A node with no children is called a leaf node.
- Balanced tree: depth (number of levels) between root and each leaf node is the same.

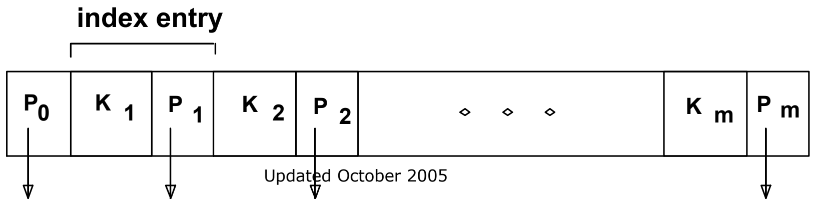
B+ Tree

- A B-tree that follows certain rules:
  - If the root is not a leaf node, it must have at least two children.
  - For a tree of order n, each node (other than root/leaf nodes) must have between n/2 and n pointers and children.
  - … (other rules)
  - Tree must always be balanced
    - Every path from the root to a leaf must have same length. 🡐 This is what matters.
    - This means that it always takes about the same time to access any record.

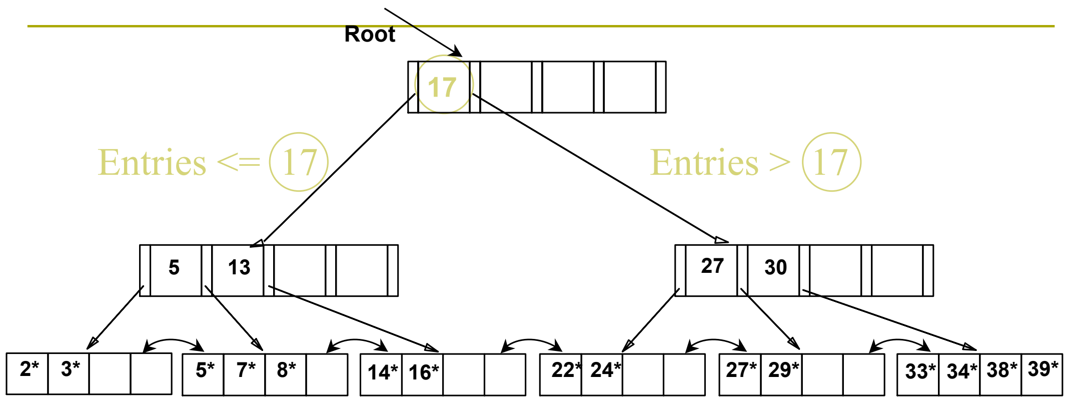
B+ Tree Indexes



- Leaf pages contain *data entries*, and are chained (prev & next)
- Non-leaf pages contain *index entries* and direct searches:



Example B+ Tree



- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

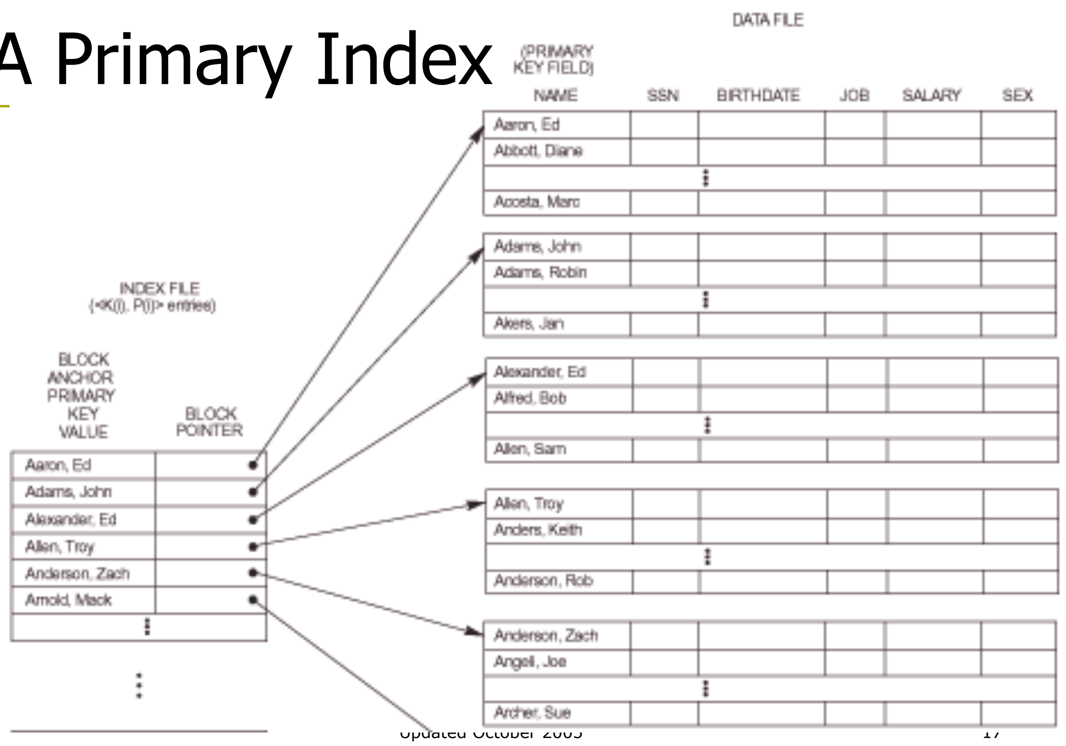
Ordered Indexes

- An ordered file that stores the index field and a pointer.
- Requires a binary search on the index file.
- Types of Ordered Indexes
  - Primary
  - Clustering
  - Secondary

Primary Indexes

- An ordered, fixed-length file that stores the primary key and a pointer.
- Each record in the index file corresponds to each block in the ordered data file.
- Sparse / nondense index

A Primary Index

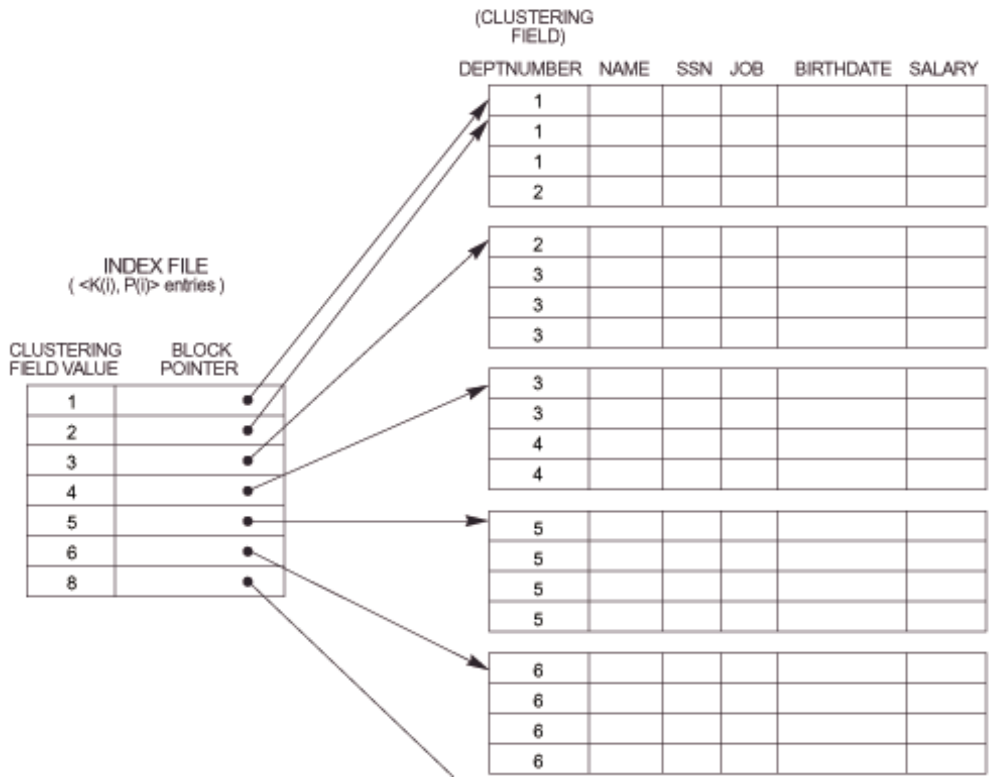


Clustering Indexes

- When the records of a data file are ordered on a non-key field, this field is called a clustering field.
- The clustering field does not have a distinct value for each record in the data file.
- A clustering index is an ordered file that stores the clustering field and a pointer.

Clustering Indexes

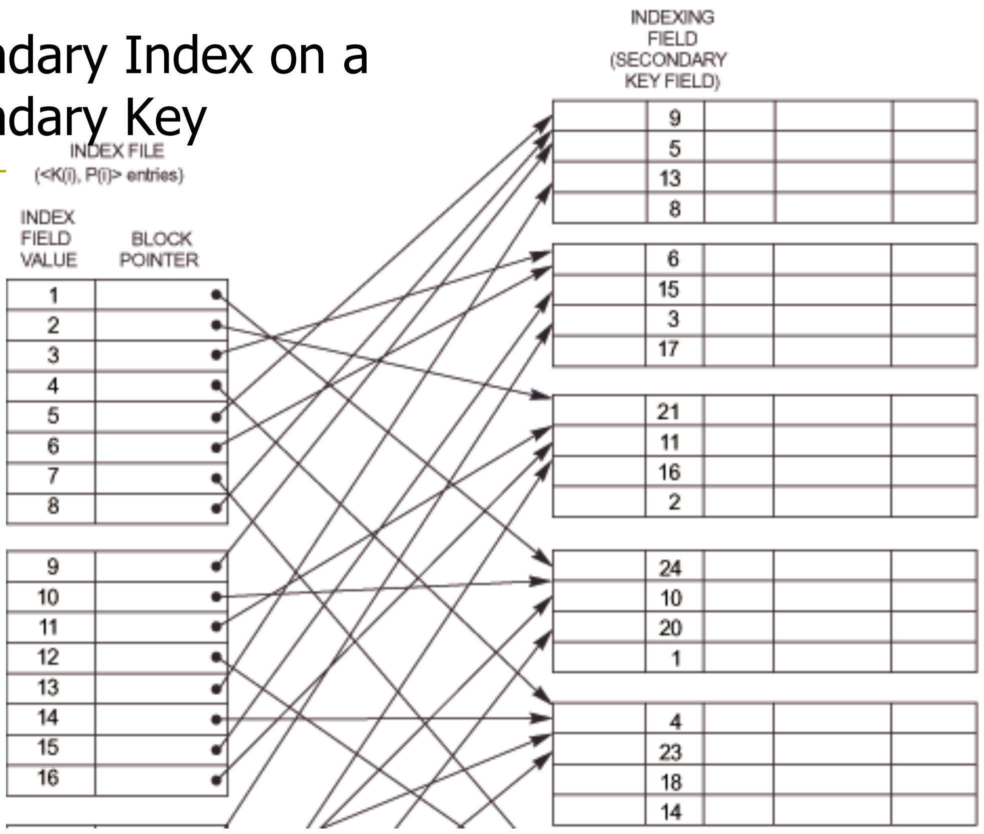
- There is one record in the index file for each distinct value of the clustering field.
- Sparse / nondense index



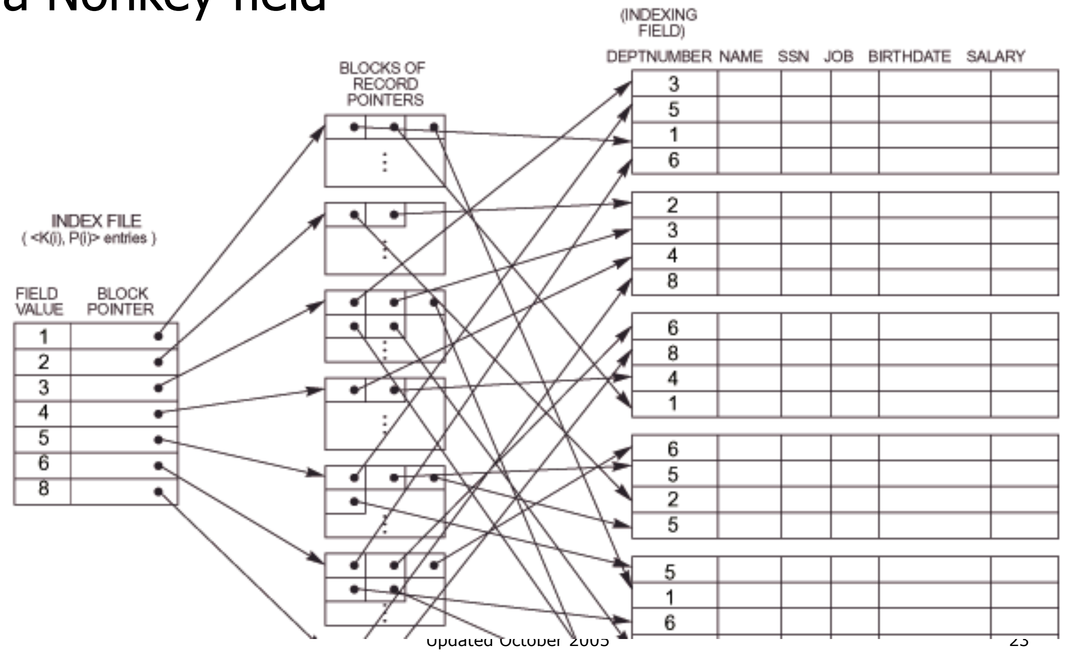
Secondary Indexes

- A secondary index is an ordered file that stores the nonordering field of a data file and a pointer.
- If the indexing field is a secondary or candidate key, then the index is dense.
- If the indexing field is a nonkey field, then the index is sparse.
- Introduces the idea of adding a level of indirection

Secondary Index on a Secondary Key



Secondary Index on a Nonkey field



Indexed Sequential Data File Organization

- Ordered data file with a multilevel primary index file based on the ordering key field.
- IBM's ISAM is an indexed sequential file organization with a two-level index.

Understanding the Workload

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- For each update in the workload:
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

Choice of Indexes

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?

Choice of Indexes (Contd.)

- One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
  - Obviously, this implies that we must understand how a DBMS evaluates queries and creates query evaluation plans!
  - For now, we discuss simple 1-table queries.
- Before creating an index, must also consider the impact on updates in the workload!
  - Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

Index Selection Guidelines

- Attributes in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
    - Clustering is especially useful for range queries; can also help on equality queries if there are many duplicates.

Index Selection Guidelines (2)

- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
  - Order of attributes is important for range queries.
  - Such indexes can sometimes enable index-only strategies for important queries.
    - For index-only strategies, clustering is not important!
- Try to choose indexes that benefit as many queries as possible. Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

Rules-of-Thumb for Indexes

- Index primary key fields
  - DBMS may do this automatically
  - Composite keys require composite indexes
- Index foreign key fields
- Index other fields frequently used in:
  - WHERE clauses
  - GROUP BY clauses
  - ORDER BY clauses
- Consider composite indexes when fields are used together in conditions

Index Method Benefits/Costs

- Heap file:
  - Efficient storage; fast scanning; fast inserts
  - Slow searches; slow deletes
- Sorted file:
  - Efficient storage; faster searches than heap
  - Slow inserts and deletes

Index Method Benefits/Costs (2)

- Clustered file:
  - Efficient storage; faster searches than heap
  - Faster inserts/deletes than sorted file
  - Only one clustering factor per file
- Unclustered tree:
  - Fast searches, fast inserts/deletes
  - Slow scans and range searches with many matches
- Hash:
  - Same as unclustered, but faster on equality searches
  - Does not support range searches

Summary

- Many alternative file organizations exist, each appropriate in some situation.
- If selection queries are frequent, sorting the file or building an index is important.
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- Index is a collection of data entries plus a way to quickly find entries with given key values.

Summary (Contd.)

- ~~Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.~~
- Can have several indexes on a given file of data records, each with a different search key.
- Indexes can be classified as clustered vs. unclustered, primary vs. secondary, and dense vs. sparse. Differences have important consequences for utility/performance.

Summary (Contd.)

- Understanding the nature of the workload for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?
- Indexes must be chosen to speed up important queries (and perhaps some updates!).
  - Index maintenance overhead on updates to key fields.
  - Choose indexes that can help many queries, if possible.
  - Build indexes to support index-only strategies.
  - Clustering is an important decision; only one index on a given relation can be clustered!
  - Order of fields in composite index key can be important.