# Physical DB Design

**Orlando Karam**

okaram@spsu.edu



SOUTHERN POLYTECHNIC STATE UNIVERSITY

# Introduction

- Need extra, detailed info about how to store tables
- Main (only?) issue is performance
- **Minimize disk access**
- Logical design should NOT change !
- Many decisions can be deferred until implementation/maintenance

# Main Decisions

- Attribute Data types
- Fields and Physical Records
- File Organization
- Indexes

# Inputs

- Normalized Relations and Attribute Definitions
- Volume estimates
- Other Issues
  - –Response time expectations
  - –Backup/Recovery/Security needs
  - –Integrity expectations
  - –DBMS etc

# Memory Hierarchy

- Many different kinds of memory
  - –Primary - RAM etc
  - –Secondary - Hard Drives, now Flash/SSD
  - –Tertiary - Removable, tapes
- Tradeoff price vs speed (for same size)
  - –So we have more of slower kinds of memory
- Also, qualitative difference
  - –Primary is volatile
  - –Secondary is permanent, non-removable
  - –Tertiary  is removable

# Fields

- Field: Smallest unit of data in db
- Issues
  - –choosing data type
  - –do we use coding, compression or encryption ?
  - –How to control integrity ?

# Hard Disks

- Most databases reside in hard disks
- Hard Disk issues
  - Info is transferred in **pages**
  - Slight difference in access time to reach each page, but considered ‘the same’
  - Speed
    - Seek time
    - Transfer rate
    - About 100 times slower than RAM
  - The less pages I read/write, the faster my database

# Choosing data type

- DBMS Specific
- Affects semantics and performance
- Use as small (in bytes) a field as possible, but needs to accommodate all possible values
- CHAR vs VARCHAR vs Text/Long
- NUMERIC vs int, float etc
- DATE, TIMESTAMP
- BLOB

# Fields: Coding

- Uses a reference instead of actual value, and puts values in table
- In a way, transforming an attribute into an entity in the ER diagram
- Saves space, needs additional lookup. slower ? faster ?
- Can improve integrity

# Fields: Controlling integrity

- Default value (DEFAULT)
- Range control (CHECK)
- Null value control (NOT NULL)
- Referential integrity
- Triggers

# Physical Records

- Record: Group of fields stored adjacently and retrieved as unit (~row)
- Page: Amount of data read or written in one IO operation (device dependent)
- Blocking factor: # records per page
  - may vary !
  - records shouldn't span 2 pages
- Smaller records mean less pages

# Denormalization

- DON'T
- Transforming normalized rels into unnormalized ones
- Possible benefits: less joins, faster
- Costs: Confusing !, wasted space, integrity threats

# Partitioning

- Horizontal: Distribute rows into several files
- Vertical: Divide rows, put each piece into a file
  - Need to replicate PK
- Combined
- Can speed up queries that require only on one partition, but queries that require data from all partitions may be slower

# Replication

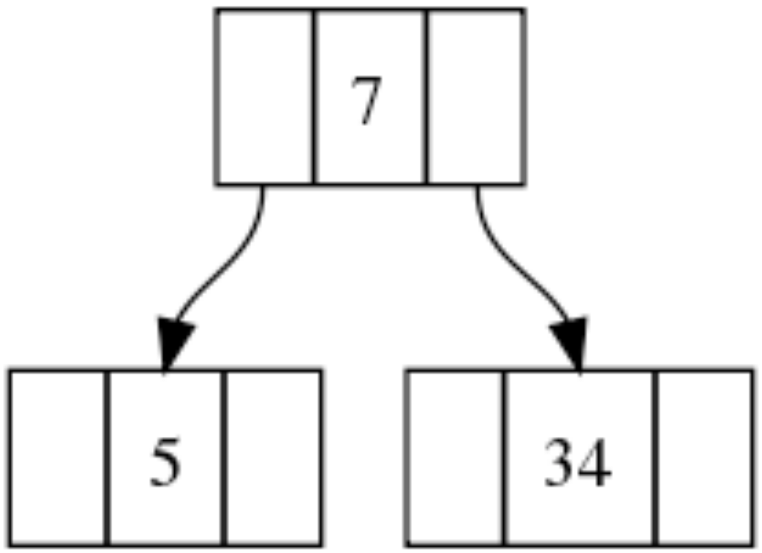
- Storing pieces of data twice
- Improves performance by minimizing contention
- Integrity problem: Duplication
- Best for data not updated often

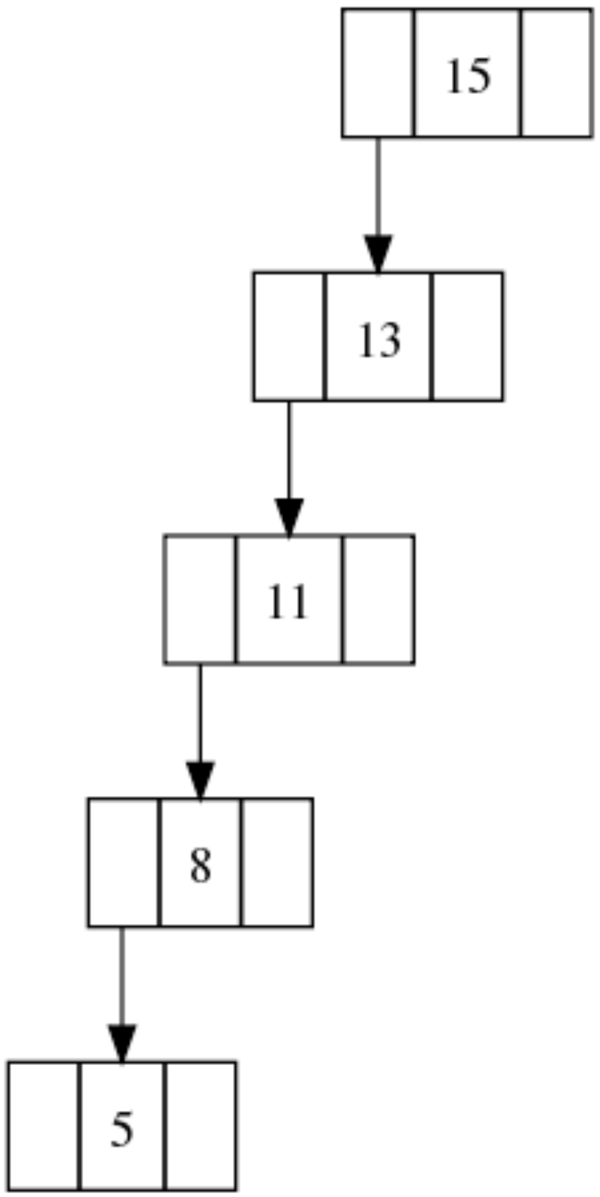
# Indexes

- Secondary data structures that improve searches
- Two main kinds
  - Search trees (and B-Trees)
  - Hash tables
- Some field or expression is the **key** for the structure
- Simplest would be to just have a list with the key and a pointer to the record
- Pointers to records would have page +offset

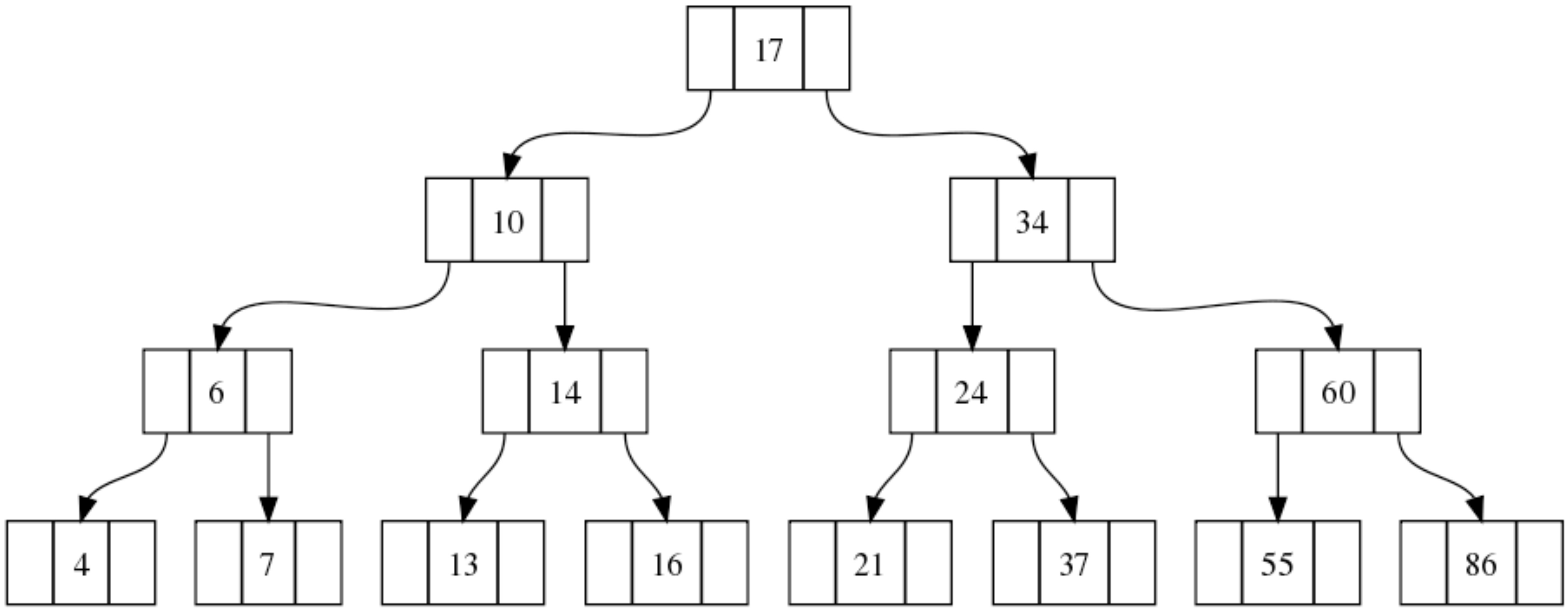
# Search Trees

- Keys (and pointers to record) are organized in a tree
- In binary tree, nodes to the left have lower keys
- Unbalanced trees don't help



# Search Trees



# Search Trees

- Balanced trees make searches way faster
- Logarithmic instead of linear
  - 10 instead of 1,000
  - 20 instead of 1,000,000
- The base (of the logarithm) is not that important
- In DBs we use node size that fits in one page

# Hash Tables

- Place records (or key and pointer) on buckets
- Use a function on the key to find the appropriate bucket (which function?)
- Problems
  - collisions (several keys map to same bucket)
  - overflow (too many keys for the bucket)
- One good solution: Expand bucket to become another hash table ... so it is a tree !!
- Most good functions help only with point queries

# Designing Physical File

- Physical file
  - Named portion of secondary memory
- Tablespace, Extent
- Constructs to link data:
  - Sequential storage
  - Pointers

# Kinds of operations (low level)

- Point query
- Range query
- Sequential Scan
- Full Scan
- Joins

# File Organization

- Techniques for physically arranging records in a file
- Main Factors
  - –Fast retrieval/throughput
  - –Space
  - –Minimize reorganization

# File Organizations

- Unordered
- Sequential / Sorted
  - Problem - reorganization
- Primary Indexed
  - Records are organized according to an index
  - May help with sequential scan (sorted)
  - ‘Wastes’ Space
- Secondary index
  - Create index for some keys (not affecting records)
- Clustered
  - Store several kinds of records together

# Indexes

- CREATE INDEX ... ON Table(field)
- Searches faster, Updates slower
- Only speed search on index keys
- When to use:
  - –large tables (50+?)
  - –unique index for PK
  - –for frequently searched fields
  - –for COUNT, GROUP etc fields

# Query Optimization

- SQL is declarative (doesn't say HOW to do the query)
- DBMS has choices
  - Reordering operations
  - Using Indexes or not
  - Joins - starting from which side
  - Different algorithms
- Modern DBMS use Cost-based optimizers
  - Generate many possibilities
  - Estimate the 'cost' of each possibility
  - Do the 'cheapest' one

# RAID

- Redundant Array of Inexpensive Disks
- Redundancy improves reliability
- Parallelism improves performance
- Levels
  - 0 - striping
  - 1 - mirroring
  - 5 parity