Files, Storage and RAID

Kathleen Durant PhD
CS 3200 Lesson 14
Northeastern University

Outline

- Review concepts from last lecture
- File organizations – Disk Manager
- Buffer manager (new content Ch. 9.3 - 9.7)
- Index organization within a file
  - Clustered vs. Non-clustered
- I/O Cost Model
- RAID (new content Ch. 9.2)

Disks and Files

- DBMS stores information on ("hard") disks.
- This has major implications for DBMS design
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully

Why Not Store Everything in Main Memory?

- *Costs too much.* $1000 will buy you either 128MB of RAM or 7.5GB of disk (as of 2005).
- *Main memory is volatile.* We want data to be saved between runs. (Obviously!)
- Typical storage hierarchy:
  - Main memory (RAM) for data currently being used.
  - Disk for the main database (secondary storage).
  - Tapes for archiving older versions of the data (tertiary storage).

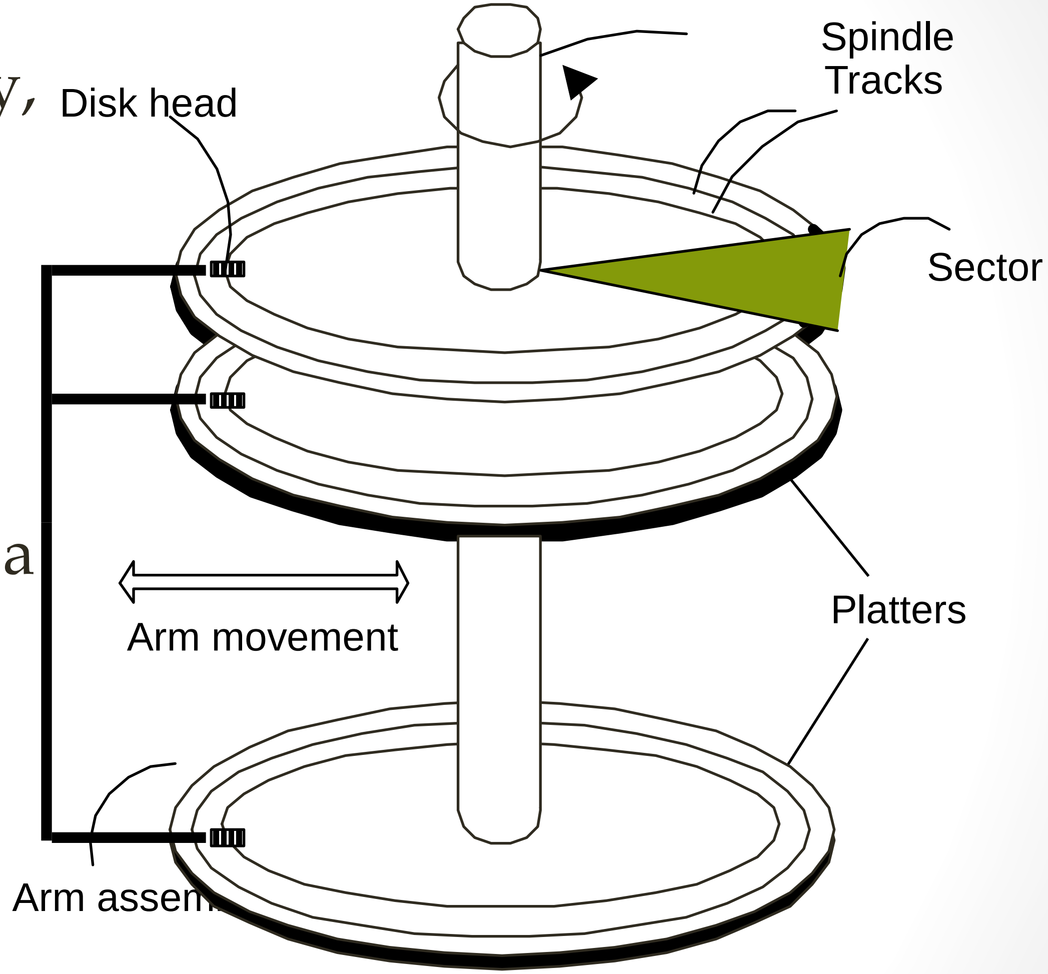
Disk Basics

- Disk: secondary storage device of choice.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
  - A disk block/page is a contiguous sequence of bytes.
  - Size is a DBMS parameter, 4KB or 8KB.
- Like RAM, disks support direct access to a page.
- Unlike RAM, time to retrieve a page varies
  - It depends upon the location on disk.

Therefore, relative placement of pages on disk has major impact on DBMS performance.

Components of a Disk

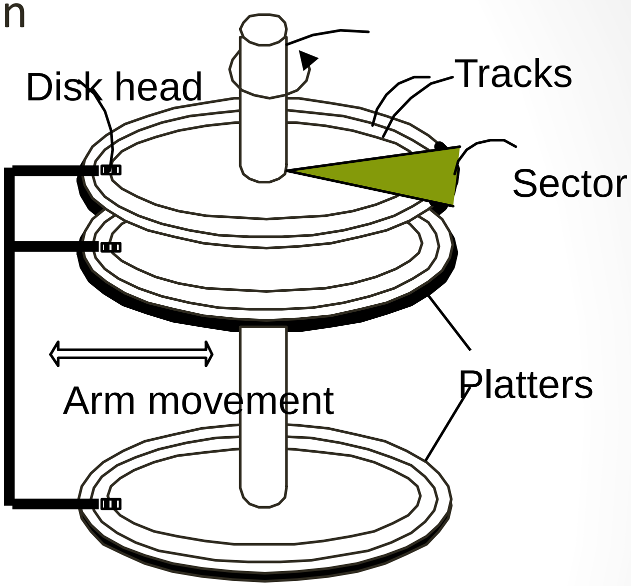
- The platters spin (say, 90rps)
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary).
- Only one head reads/writes at any one time.



- *Block size* is a multiple of a *sector size* (which is fixed).

Accessing a Disk Page

- Time to access (read/write) a disk block:
  - *Seek time:* moving arms to position disk head on track
  - *Rotational delay:* waiting for block to rotate under head
  - *Transfer time:* actually moving data to & from disk surface
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost:



Reduce seek & rotation delays

Arranging Pages on Disk

- Blocks in a file should be arranged sequentially on disk (by `next'), to minimize seek and rotational delay.
- *`Next'* block concept:
  - Blocks on same track, followed by
  - Blocks on same cylinder, followed by
  - Blocks on adjacent cylinder
- For a sequential scan, *pre-fetching* several pages at a time is a big win

Disk Space Management

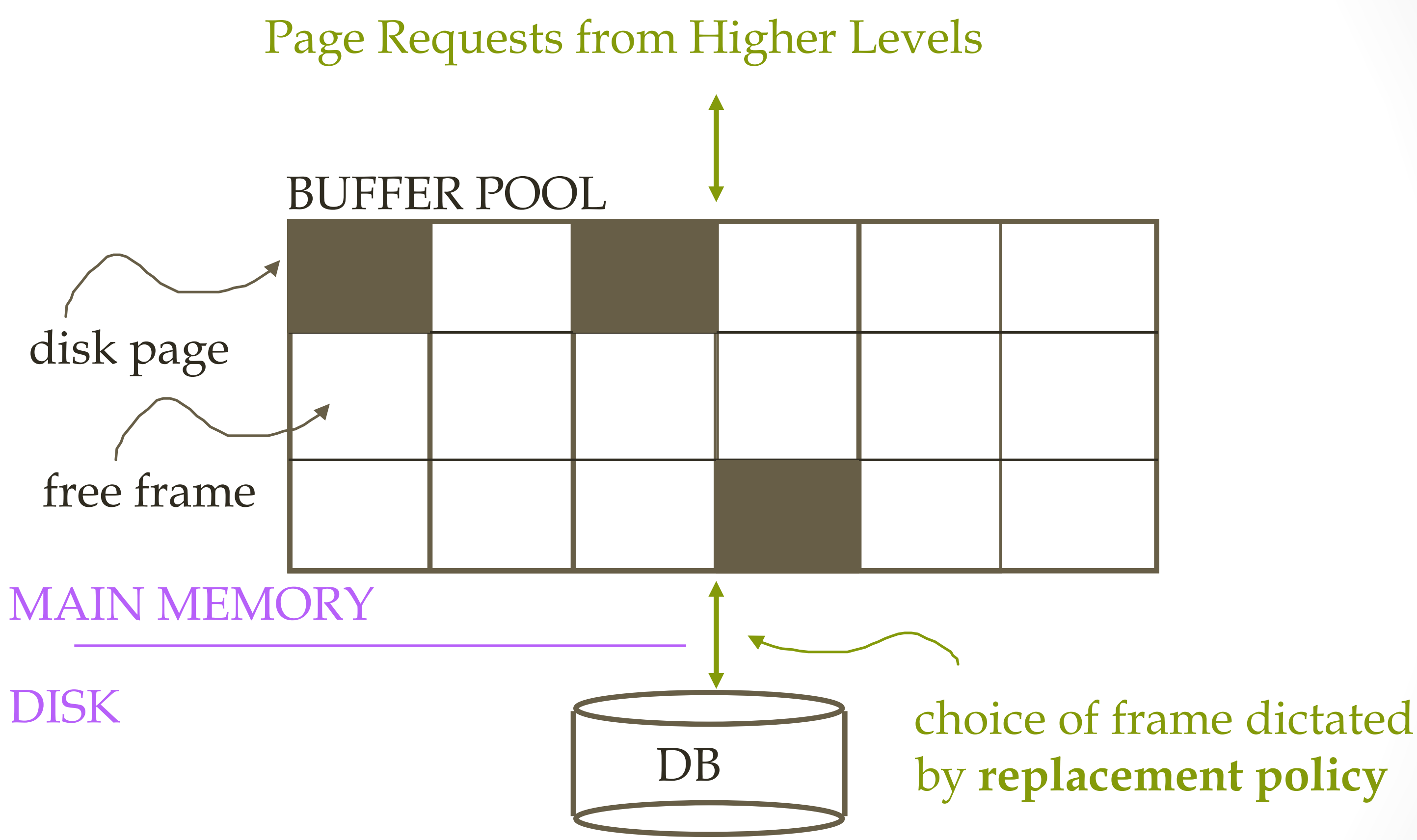
- Lowest layer of DBMS software manages space on disk.
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk. Higher levels don't need to know how this is done, or how free space is managed.

When a Page is Requested ...

- If requested page is not in the buffer pool:
  - Choose a frame for *replacement*
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- *Pin* the page and return its address.

If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
- *Table of <frame#, pageid> pairs is maintained.*

Buffer Management Activities

- Requestor of page must unpin it, and indicate whether page has been modified:
  - *Dirty* bit is used for this.
- Page in pool may be requested multiple times
  - A *pin count* is used. A page is a candidate for replacement iff *pin count* = 0.
- Concurrency control & recovery manager will handle the additional I/O when a frame is chosen for replacement. (*Write-Ahead Log protocol*)

Buffer Replacement Policy

- Frame is chosen for replacement by a *Replacement policy:*
  - Least-recently-used (LRU), Clock, MRU, FIFO, LIFO etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

Data usage patterns

- Just a few basic data access patterns in RDBS, with **noticeable locality behaviors**
  - Reason why stochastic policies do not work well in managing buffer
- DB operations can be broken down or decomposed into a subset of these access patterns
- To reduce I/Os expose these patterns to the buffer manager for correct estimation of:
  - Buffer size - many queries share the buffer pool; need to know how to allocate frames to each query
  - Replacement policy - evict unused pages to make room for newly requested pages.
    - Steal or no-steal policy

Disk Space Manager

- Disk space manager is the lowest layer of DBMS managing space on disk.
- Higher levels call upon this layer to:
  - Allocate/de-allocate a page or sequence of pages
  - Read/Write a page
- Requests for a sequence of pages are satisfied by *allocating the pages sequentially* on disk!
  - Higher levels don't need to know how this is done, or how free space is managed.

File Control? DBMS vs. OS System

Discussion: Operating System already knows how to manage disk space

Why not let OS manage these tasks?

- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - Pin a page in buffer pool, force a page to disk (important for implementing Concurrency Control & Recovery),
  - Adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.
- Too important to the efficiency of a DBMS to leave it to another system

Record Abstraction: File of Records

- Page or block is OK when doing I/O, but higher levels of DBMS operate on ***records***, and *files of records*.
- **FILE:** A collection of pages, each file containing a specific collection of records. Must support:
  - Insert/Delete/Modify record
  - Read a specific record (specified using a *record id*)
  - Scan all records (possibly with some conditions on the records to be retrieved)

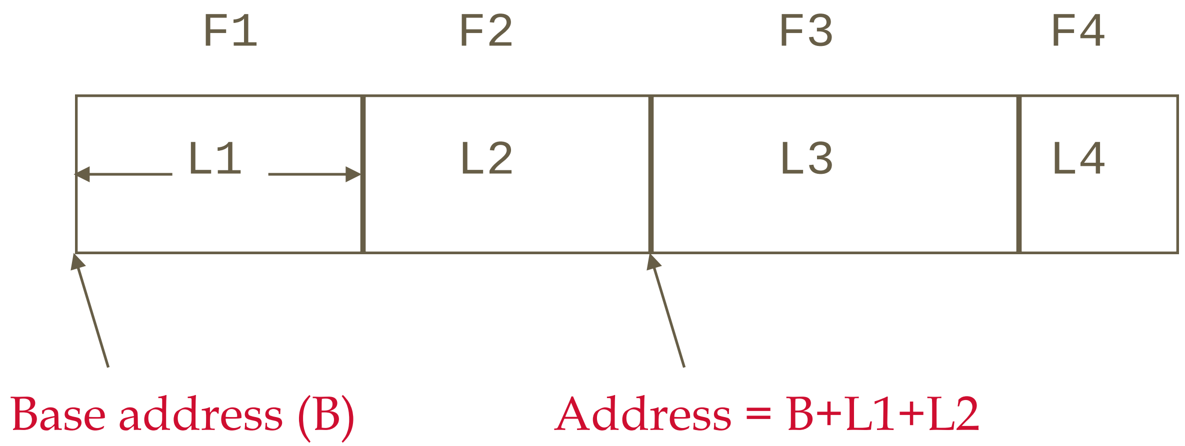
Files

- Access method layer offers an abstraction of disk-resident data: a file of records residing on multiple pages
  - A number of *fields* are organized in a *record*
  - A collection of records are organized in a *page*
  - A collection of pages are organized in a *file*

File structure types

- Heap (random order) files
  - Suitable when typical access is a file scan retrieving all records.
- Sorted Files
  - Best if records must be retrieved in some order, or only a `range' of records is needed.
- Indexes = data structures to organize records via trees or hashing.
  - Like sorted files, they speed up searches for a subset of records, based on values in certain ("search key") fields
  - Updates are much faster than in sorted files.

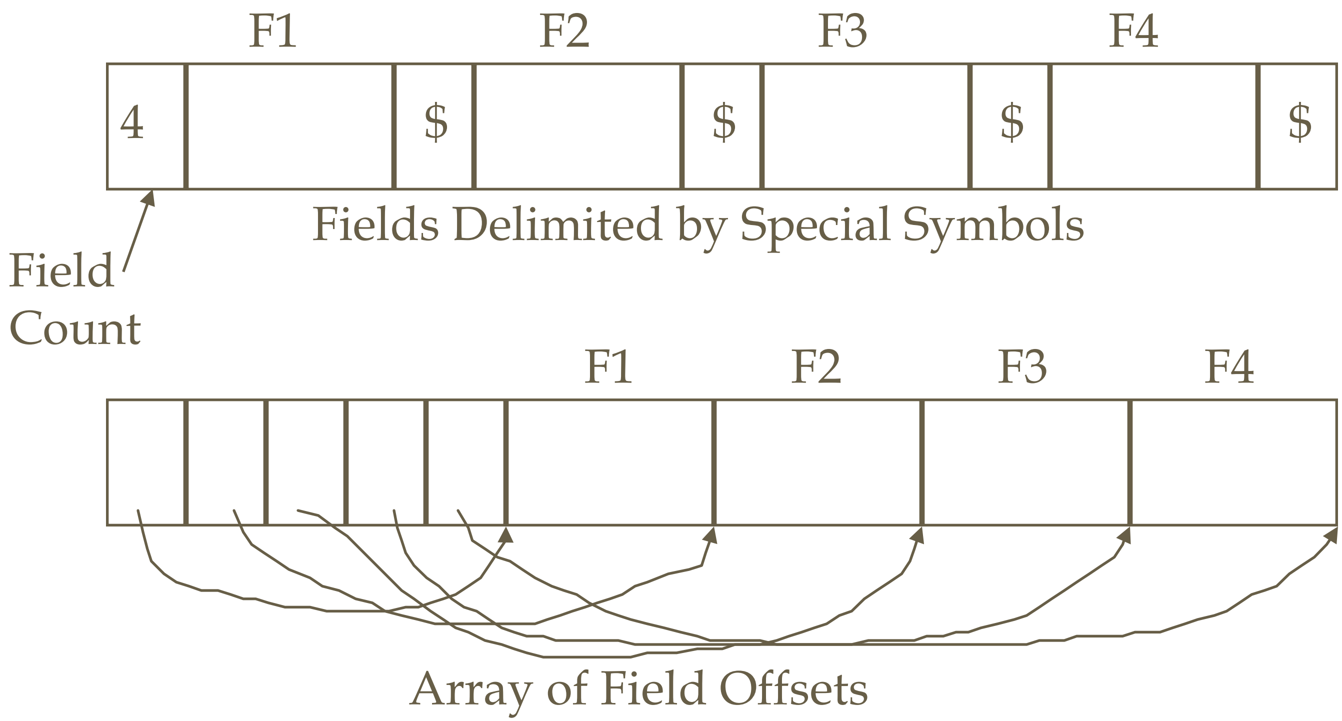
Record Formats: Fixed Length



- Information about field types same for all records in a file; stored in *system catalogs.*
- Finding *i'th* field requires scan of record.

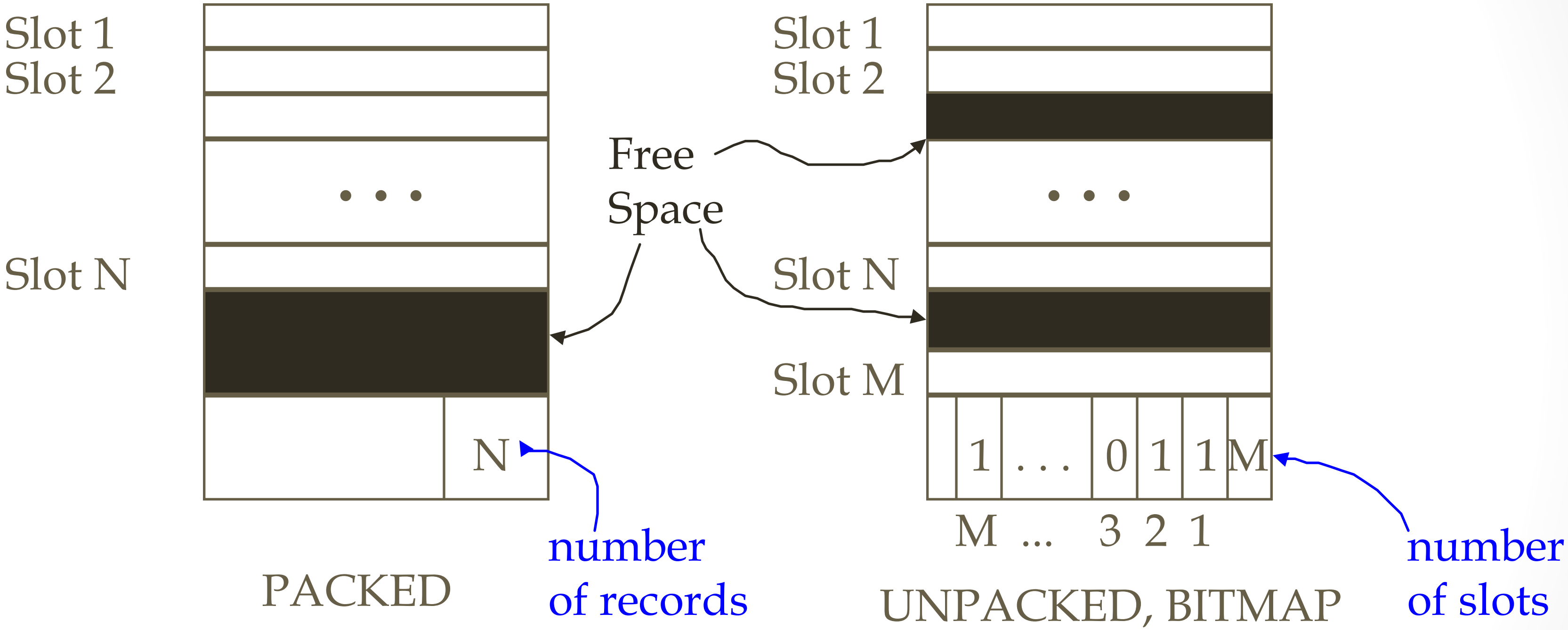
Record Formats: Variable Length

Two alternative formats (# fields is fixed):



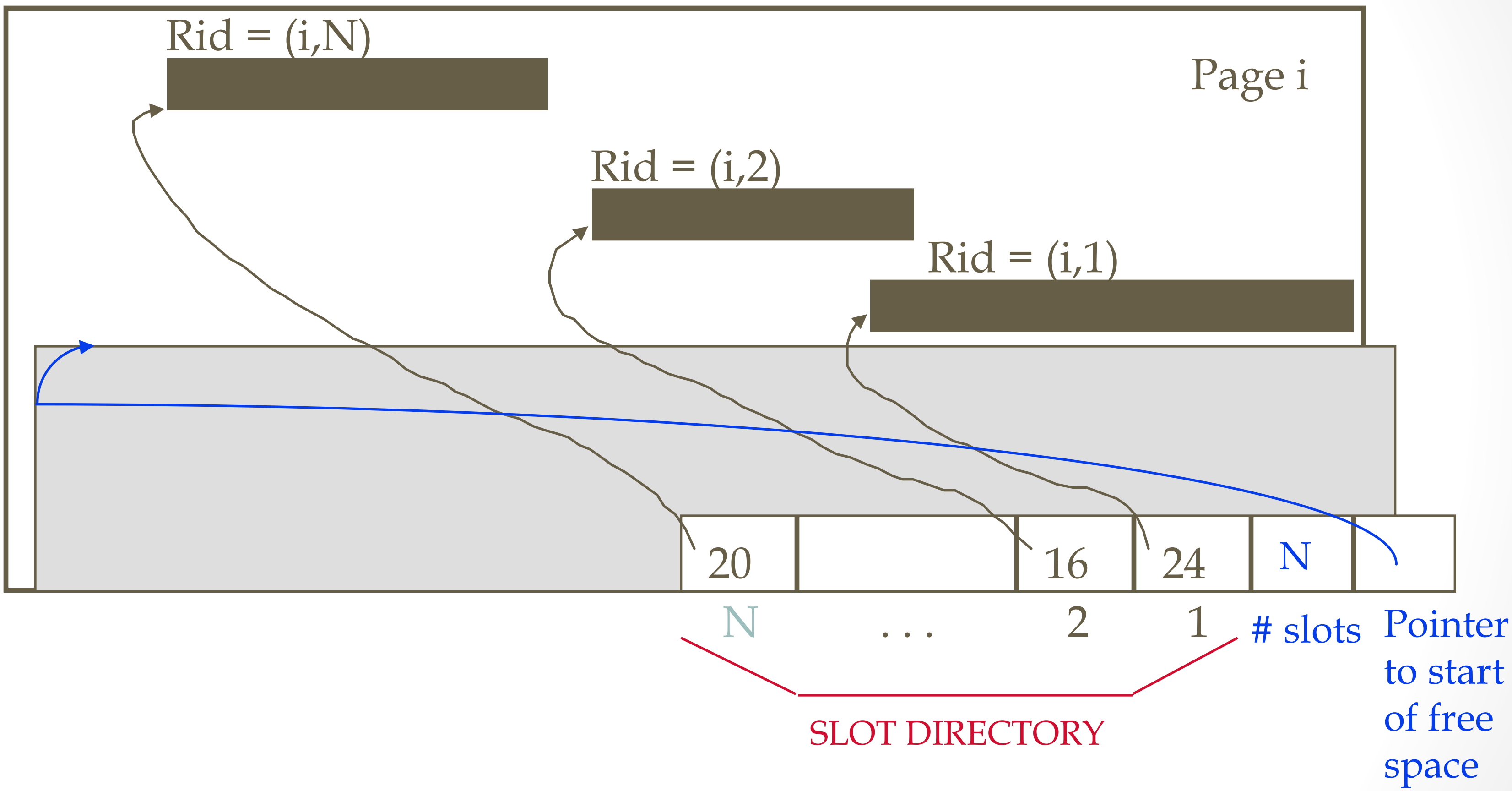
Second offers direct access to i'th field, efficient storage of nulls (special *don't know* value); small directory overhead.

Page Formats: Fixed Length Records



Record id = <page id, slot #>. In first alternative, move a record for free space management – involves updating rid; may not be acceptable.

Page Formats: Variable Length Records

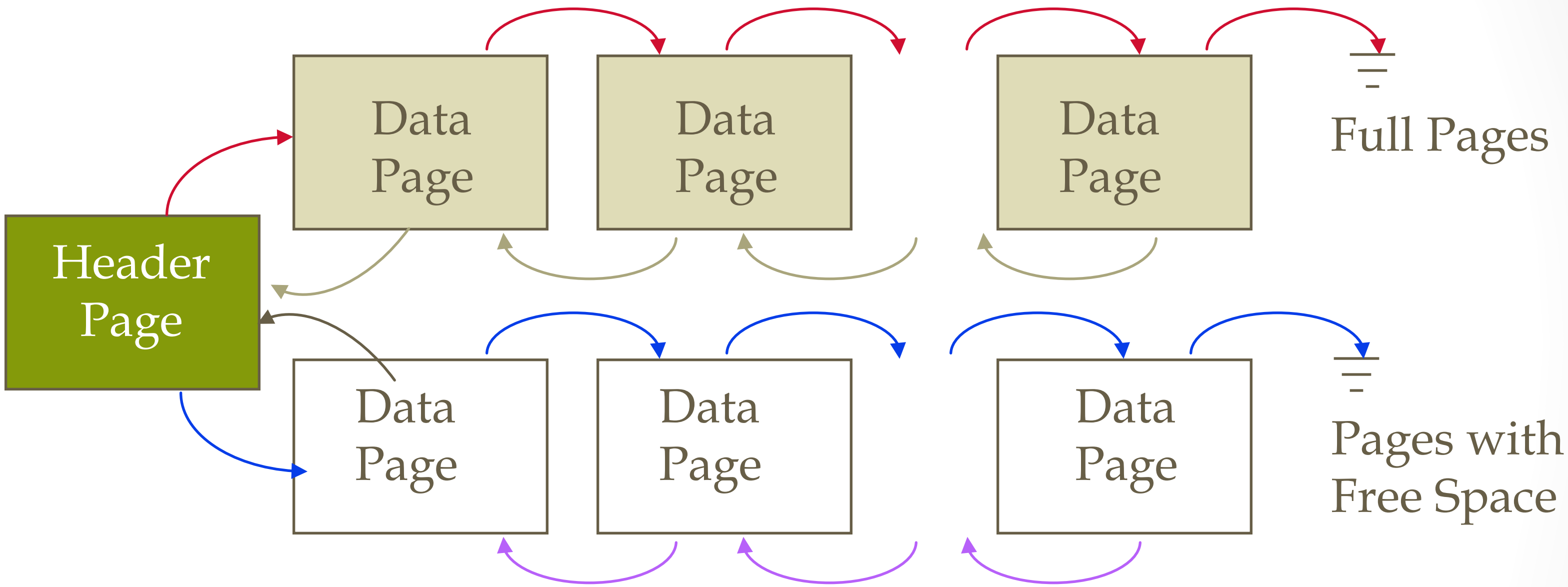


Can move records on page without changing rid; so, attractive for fixed-length records too.

Unordered (Heap) Files

- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - Keep track of the *pages* in a file
  - Keep track of *free space* on pages
  - Keep track of the *records* on a page
- There are many alternatives for keeping track of this.

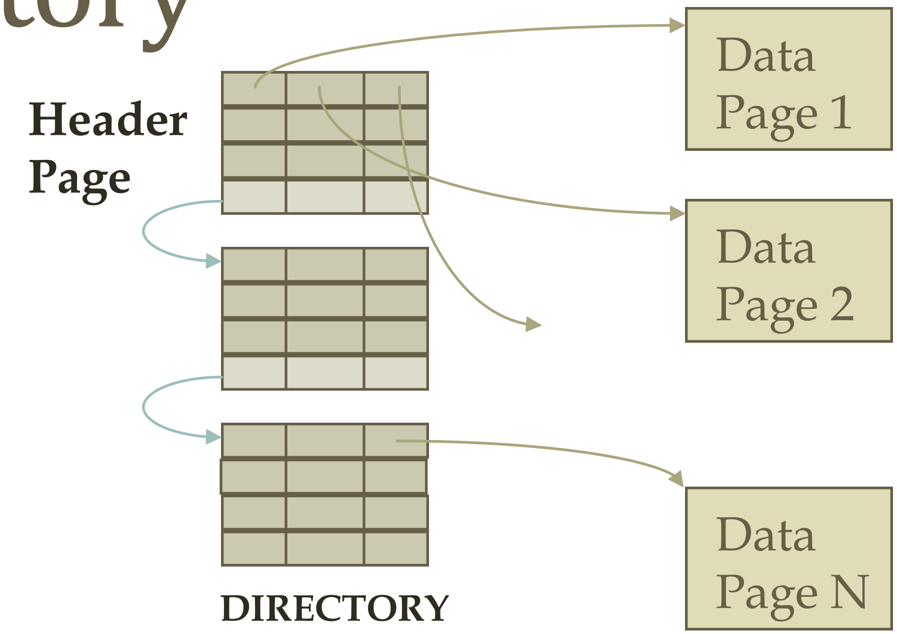
Heap File Implemented as a List



The header page id and Heap file name must be stored someplace.

Each page contains 2 `pointers' plus data.

Heap File Using a Page Directory



- The entry for a page includes a pointer to the page and can include the number of free bytes on the page.
- The directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all Heap File pages*!

Indexes

- A Heap file allows us to retrieve records:
  - by specifying the *rid*, or
  - by scanning all records sequentially
- Sometimes, we want to **retrieve records by specifying the *values in one or more fields***
  - ***Examples:***
  - Find all students in the "CS" department
  - Find all students with a gpa > 3
- Indexes are file structures that enable us to answer such value-based queries efficiently.

Indexes

- An index on a file speeds up selections on the search key fields for the index
  - Any subset of the fields of a relation can be the search key for an index on the relation
  - Search key is not the same as a key in the DB
- An index contains a collection of data entries, and supports efficient retrieval of all data entries k* with a given key value k.

Cost Model Analysis Review

- We ignore CPU costs, for simplicity:
  - B: The number of data pages (Blocks)
  - R: Number of records per page (Records)
  - D: (Average) time to read or write a single disk page
- Measuring number of page I/O's
  - ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated
- Average-case analysis; based on several simplifying assumptions

Far from Precise but Good enough to show the overall trends!

Comparing File Organization

- Heap files (random order; insert at eof)
- Sorted files, sorted on attributes <age, sal>
- Clustered B+ tree file, Alternative 1, search key <age, sal>
- Heap file with unclustered B + tree index on search key <age, sal>
- Heap file with unclustered hash index on search key <age, sal>

Five operations to compare

- Scan: Fetch all records from disk
- Equality search
- Range selection
- Insert a record
- Delete a record

Assumptions for the File Organizations

- Heap Files:
  - Equality selection on key; exactly one match.
- Sorted Files:
  - Files compacted after deletions.
- Indexes:
  - Alternatives 2, 3: data entry size = 10% of record size
- Tree: 67% occupancy (Close to AUC for 1 std dev.).
  - Implies file size = 1.5 data size (because of extra free space)
- Hash: No overflow buckets.
  - 80% page occupancy => File size = 1.25 data size

Summary of workload

File Type	Scan	Equality Search	Range Search	Insert	Delete
Heap	BD	.5BD	BD	2D	Search + D
Sorted	BD	D $\log_2 B$	D$\log_2 B$ + # matching p.	Search + BD	Search + BD
Clustered	1.5BD	D $\text{Log}_F 1.5B$	D$\text{Log}_F 1.5B$ + **# matched pages**	Search + D	Search + D
Unclustered tree index	BD(R + 0.15)	D(1+ $\log_F 0.15B$)	D($\text{Log}_F 0.15B$ + **# matching records**)	D(3 + $\log_F 0.15B$)	Search + 2D
Unclustered Hash index	BD(R + 0.125)	2D	BD	4D	Searches + 2D

RAID

- Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
  - High capacity and high speed by using multiple disks in parallel
  - High reliability by storing data redundantly, so that data can be recovered even if a disk fails
- Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks -> more failures. Redundant information allows reconstruction of data if a disk fails.

New Problems from RAID

- The chance that some disk out of a set of *N* disks will fail is much higher than the chance that a specific single disk will fail.
  - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)

MTTF = Mean time to failure

Improvement of Reliability via Redundancy

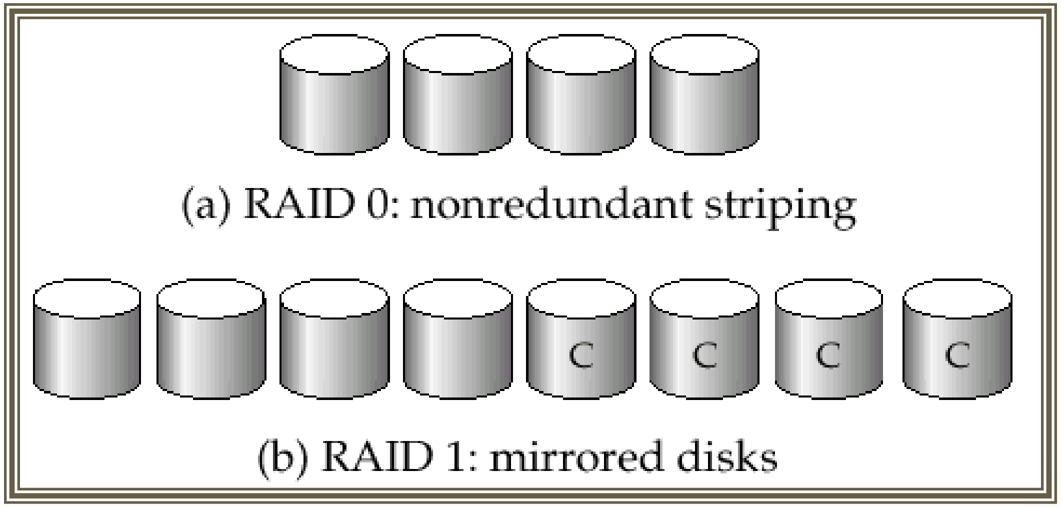
- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or **shadowing**)
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - Reads can take place from either disk
  - If one disk in a pair fails, data still available in the other
    - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
        - Except for dependent failure modes such as fire or building collapse or electrical power surges

Improvement in Performance via Parallelism

- Two main goals of parallelism in a disk system:
  1. Load balance multiple small accesses to increase throughput
  2. Parallelize large accesses to reduce response time.
- Improve transfer rate by striping data across multiple disks.
- **Bit-level striping** – split the bits of each byte across multiple disks
  - But seek/access time worse than for a single disk
    - Bit level striping is not used much any more
- **Block-level striping** – with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1
  - Requests for different blocks can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel

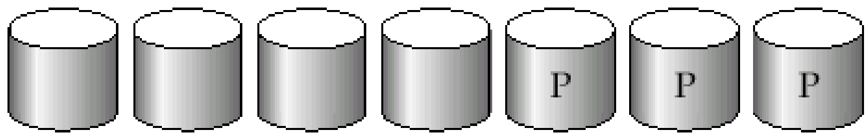
RAID Levels 0,1

- RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics
- **RAID Level 0**: Block striping; non-redundant.
  - Used in high-performance applications where data lost is not critical.
  - Best write performance of all RAID levels
- **RAID Level 1**: Mirrored disks
  - Most expensive solution
  - Maximum transfer rate = transfer rate of one disk
  - Popular for applications such as **storing log files in a database system**
  - **Each write is 2 writes since has 2 copies of the data**
  - **Read is scheduled for the copy that has the lowest expected wait time**
- **RAID 0+1**: Mirrored disks with block striping
  - Offers best write performance.
  - Maximum transfer rate = transfer rate of one disk



RAID Level 2

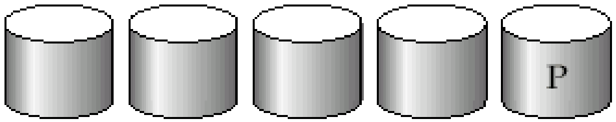
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.
  - Striping unit is a single bit
  - Parallel reads, a write involves two disks.
  - Maximum transfer rate = aggregate bandwidth
  - Good for large data requests since block size defined across all disks - but bad for small requests
  - Number of parity bits grows logarithmically with number of data disks
  - Parity data uses Hamming code - contains quality of data and quality of disks



(c) RAID 2: memory-style error-correcting codes

RAID Level 3

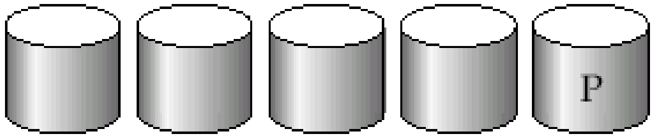
- **RAID Level 3**: Bit-Interleaved Parity
  - Striping Unit: One bit.
  - Only 1 check disk so lowest overhead
  - Each read and write request involves all disks; disk array can process one request at a time.
  - Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O. Performance similar to RAID2
  - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
  - To recover data of a damaged disk, compute XOR of bits from other disks (including parity bit disk)



(d) RAID 3: bit-interleaved parity

RAID Levels 4

- **RAID Level 4:** Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.
  - Striping Unit: One disk block. One check disk.
  - Parallel reads possible for small requests (can limit request to the disk where the data resides) , large requests can utilize full bandwidth
  - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.



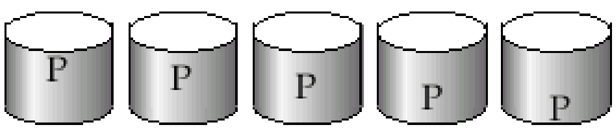
(e) RAID 4: block-interleaved parity

RAID Levels (4 Cont.)

- Provides higher I/O rates for independent block reads than Level 3
  - Block read goes to a single disk, so blocks stored on different disks can be read in parallel
- Before writing a block, parity data must be computed
  - Can be done by using old parity block, old value of current block and new value of current block (2 block reads + 2 block writes)
  - Or by recomputing the parity value using the new values of blocks corresponding to the parity block
    - More efficient for writing large amounts of data sequentially
- Parity block becomes a bottleneck for independent block writes since every block write also writes to parity disk

RAID Level 5

- **RAID Level 5:** Block-Interleaved Distributed Parity; partitions data and parity among all *N* + 1 disks, rather than storing data in *N* disks and parity in 1 disk.
  - E.g., with 5 disks, parity block for *n*th set of blocks is stored on disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks.
  - Higher I/O rates than Level 4.
    - Block writes occur in parallel if the blocks and their parity blocks are on different disks.
    - Subsumes Level 4: provides same benefits, but avoids bottleneck of parity disk.

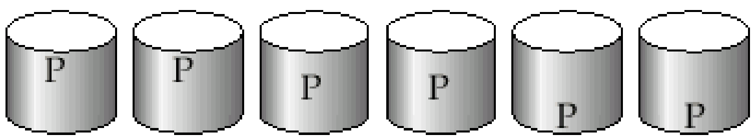


(f) RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID Level 6

- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
- Can recover from 2 simultaneous disk failures
  - Better reliability than Level 5 at a higher cost; not used as widely.



(g) RAID 6: P + Q redundancy

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - E.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used since bit-striping forces single block reads to access all disks, wasting disk arm movement
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications
- So competition is mainly between 1 and 5

Choice of RAID Level (Cont.)

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased at a slower rate (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - So there is often no extra monetary cost for Level 1
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

Summary: File and Buffer Manager

- Disks provide cheap, non-volatile storage.
  - Random access, but cost depends on location of page on disk; important to arrange data sequentially to minimize *seek* and *rotation* delays.
- Buffer manager brings pages into RAM.
  - Page stays in RAM until released by requestor.
  - Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
  - Choice of frame to replace based on *replacement policy.*
  - Tries to *pre-fetch* several pages at a time.

Summary: File Organization

- File layer keeps track of pages in a file, and supports abstraction of a collection of records.
  - Pages with free space identified using linked list or directory structure (similar to how pages in file are kept track of).
- Indexes support efficient retrieval of records based on the values in some fields.
- Many alternatives file organizations exists, each appropriate in some situations

Summary: File manager

- DBMS vs. OS File Support
  - DBMS needs features not found in many OS's, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.
- Variable length record format with field offset directory offers support for direct access to i'th field and null values.
- Slotted page format supports variable length records and allows records to move on page.

Summary: Index

- Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
- Choice orthogonal to indexing technique used to locate data entries with a given key value.
  - 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 and primary vs. secondary.
- Differences have important consequences for utility/performance.

Summary: Workload to Index

- Understanding the nature of the workload and performance goals essential to developing a good design.
  - What are the important queries and updates?
  - What attributes and 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; since only one index on a given relation can be clustered
  - Order of fields in composite index key can be important.