Dept. of Computer Science and Engineering
University of Rajshahi
www.ru.ac.bd

Dr. Shamim Ahmad

Chapter 11: Storage and File Structure

- Brief overview of Physical Storage Media for Databases
  - To know its incidence on the design and usage of DBMSs
  - Magnetic Disks
  - RAID
  - Storage Access and buffer management
- File Organization
  - Representation of records
  - Organization of Records in Files
  - Data-Dictionary Storage
- Storage and File Organization in Oracle 10g

Classification of Physical Storage Media

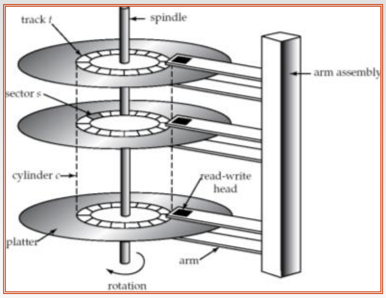
- In the end, a database must be physically stored in computer(s)
- Several aspect of storage media must be taken into account
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability
    - data loss on power failure or system crash
    - physical failure of the storage device
  - Can differentiate storage into:
    - **volatile storage:** loses contents when power is switched off
    - **non-volatile storage**:
      - Contents persist even when power is switched off.
      - Includes secondary and tertiary storage, as well as batter- backed up main-memory.

Physical Storage Media for DBMSs

- **Main memory**:
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
  - **Volatile** — contents of main memory are usually lost if a power failure or system crash occurs.
- **Magnetic-disk**
  - Primary medium for the long-term non-volatile online storage of data; typically stores entire database.
  - Data must be moved from disk to main memory for access, and written back for storage
    - Much slower access than main memory
  - **direct-access** – possible to read data on disk in any order
  - Reliable: usual mean time to failure is now of **about 3 to 5 years**
- **Optical and tape storage**
  - Mainly backups (offline storage)

Magnetic Hard Disk Mechanism

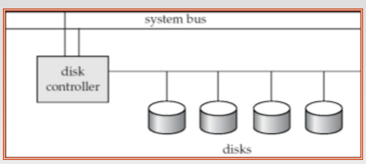
- It is worth taking a look at how magnetic disks work.
  - After all they are the place where the databases are stored!



Magnetic Disks

- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information.
- Surface of platter divided into circular **tracks**
  - Over **50K-100K** tracks per platter on typical hard disks
- Each track is divided into **sectors.**
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically **512 bytes**
  - Typical sectors per track: **500** (on inner tracks) to **1000** (on outer tracks)
- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - multiple disk platters on a single **spindle (1 to 5 usually)**
  - one head per platter, mounted on a common arm.
- **Cylinder** *i* consists of i^{th} track of all the platters

Disk Subsystem



- Multiple disks connected to a computer system through a controller
  - **Controllers functionality (checksum, bad sector** remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
  - **ATA (AT adaptor)** range of standards
  - **SATA (Serial ATA)**
  - **SCSI (Small Computer System Interconnect)** range of standards
  - Several variants of each standard (different speeds and capabilities)

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write **request is issued** to when data transfer begins. Consists of:
  - **Seek time** – time it takes to **reposition the arm over the correct track.**
    - Average seek time is **1/2 the** worst case seek time.
    - **4 to 10 milliseconds** on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
    - Average latency is 1/2 of the worst case latency.
    - **4 to 11 milliseconds** on typical disks (**5400 to 15000 r.p.m.)**
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - **25 to 100 MB per** second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. **ATA-5: 66 MB/sec, SATA: 150 MB/sec, Ultra 320 SCSI: 320 MB/s**
    - **Fiber Channel (FC2Gb): 256 MB/s**
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
  - Nowadays, typically **3 to 5 years**

Optimization of Disk-Block Access

- **Block** – a contiguous sequence of sectors from a **single track**
  - data is transferred between disk and main memory in blocks
  - sizes range **from 512 bytes to several kilobytes**
    - **Smaller blocks**: more transfers from disk
    - **Larger blocks:** more space wasted due to partially filled blocks
    - Typical block sizes today range **from 4 to 16 kilobytes**
  - We'll see how this is important for database storage structure
- **Disk-arm-scheduling** algorithms order pending **accesses to tracks** so that disk arm movement is minimized
  - **elevator algorithm** : move disk arm in one direction (from outer to inner tracks or vice versa), **processing next request** in that direction, till **no more requests** in that direction, then reverse direction and repeat

Optimization of Disk Block Access (Cont.)

- **File organization** – **optimize block access** time by organizing the blocks to correspond to how data will be accessed
  - E.g. Store related information on **the same or nearby cylinders**.
  - Files may get **fragmented** over time
    - E.g. if data is **inserted to/deleted** from the file
    - Or free blocks on disk are **scattered,** and newly created file has its blocks scattered over the disk
    - **Sequential access to a fragmented** file results in increased disk arm movement
  - Some systems have utilities to **defragment** the file system, in order to speed up file access

RAID

- The choice of **disk structure** is very important in databases. Important factors, besides price, are:
  - **Capacity**
  - **Speed**
  - **Reliability**
- **RAID: Redundant Arrays of Independent Disks**
  - disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - high capacity and high speed by using multiple disks in parallel, and
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails

Improvement of Reliability via Redundancy

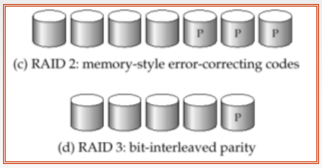
- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- E.g., **Mirroring** (or **shadowing**) – **RAID level 1**
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - Reads can take 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)
- Mean time to data loss depends on mean time to failure, and mean time to repair

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
  - In an array of eight disks, write bit ***i*** **of each byte to disk *i*.**
  - Each access can read **data at eight times the rate of a single disk.**
  - 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**
  - Usually called **RAID level 0**

RAID Levels 2 and 3

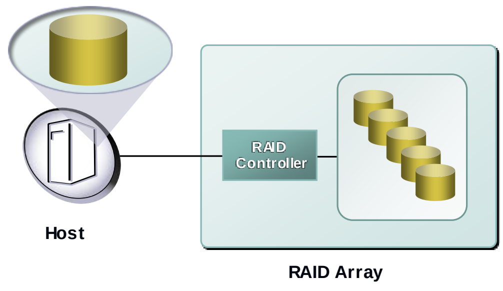
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.
- **RAID Level 3**: Bit-Interleaved Parity
  - a single parity bit is enough for error correction, not just detection, since we know which disk has failed
    - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
    - To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)
- RAID levels 4-6 elaborate on this idea of parity



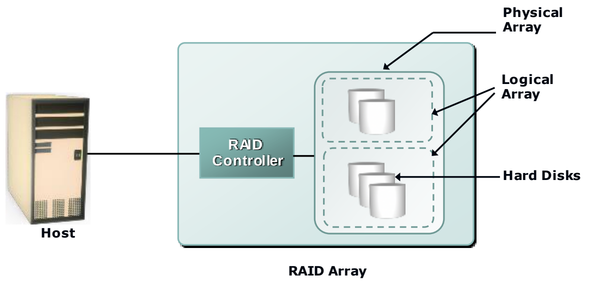
Choice of RAID Level

- Mirroring provides much better write performance than Parity
  - For Parity read operations are needed before a write, whereas Mirroring only requires 2 writes
  - Mirroring preferred for high update environments such as log disks
- Mirroring needs more disks for the same capacity
  - Higher cost per capacity unit
    - But price of storage capacity is already low and decreasing
- Nowadays, for database systems
  - (Distributed) Parity, with RAID Level 5, is preferred for applications with low update rate, and large amounts of data
  - Mirroring is preferred for all other applications
- The choice of the storage structure is an important first step when designing a real database!

RAID – Redundant Array of Independent Disks



RAID Array Components



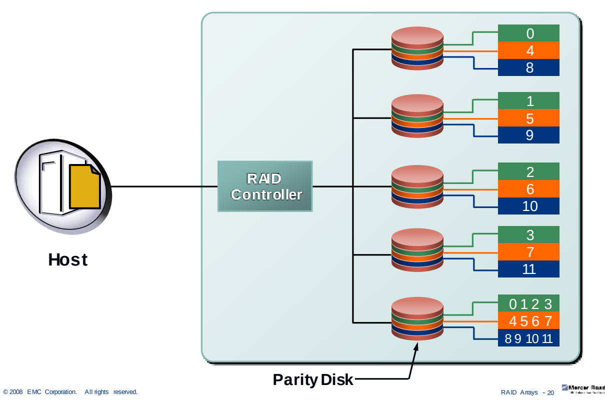
RAID Implementations

- Hardware (usually a specialized disk controller card)
  - Controls all drives attached to it
  - Array(s) appear to host operating system as a regular disk drive
  - Provided with administrative software
- Software
  - Runs as part of the operating system
  - Performance is dependent on CPU workload
  - Does not support all RAID levels

RAID Levels

- 0 Striped array with no fault tolerance
- 1 Disk mirroring
- 3 Parallel access array with dedicated parity disk
- 4 Striped array with independent disks and a dedicated parity disk
- 5 Striped array with independent disks and distributed parity
- 6 Striped array with independent disks and dual distributed parity
- Nested RAID (i.e., 1 + 0, 0 + 1, etc.)

RAID Redundancy: Parity



Parity Calculation

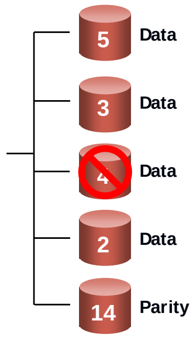
5 + 3 + 4 + 2 = 14

The middle drive fails:

5 + 3 + ? + 2 = 14

? = 14 – 5 – 3 – 2

? = 4

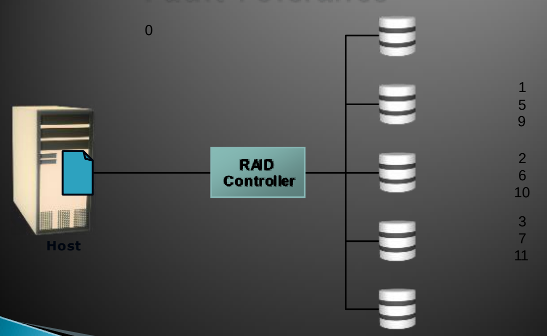


RAID Array

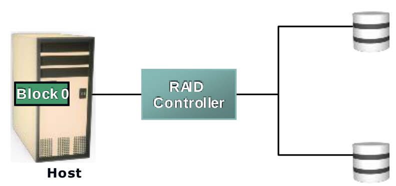
Lecture 8, 9, 10

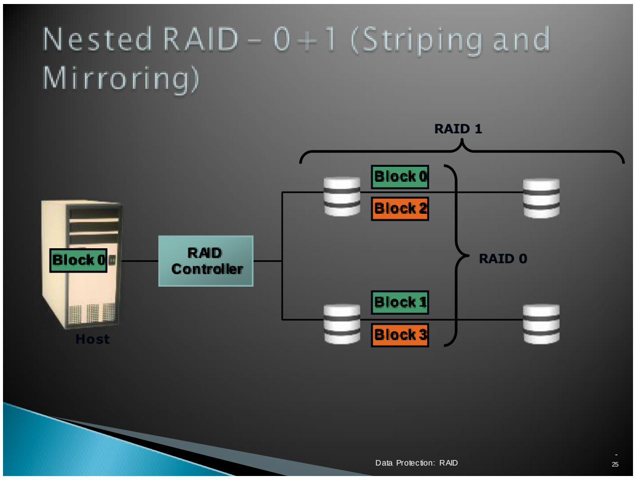
- Different RAID levels and their suitability for different application environments: RAID 0, RAID 1

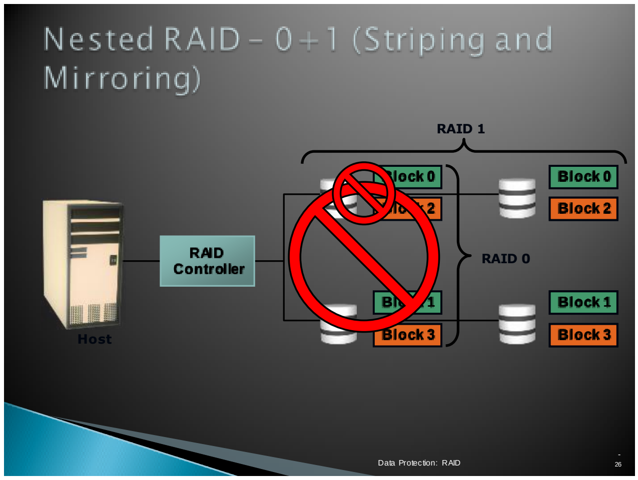
RAID 0 – Striped Array with no Fault Tolerance

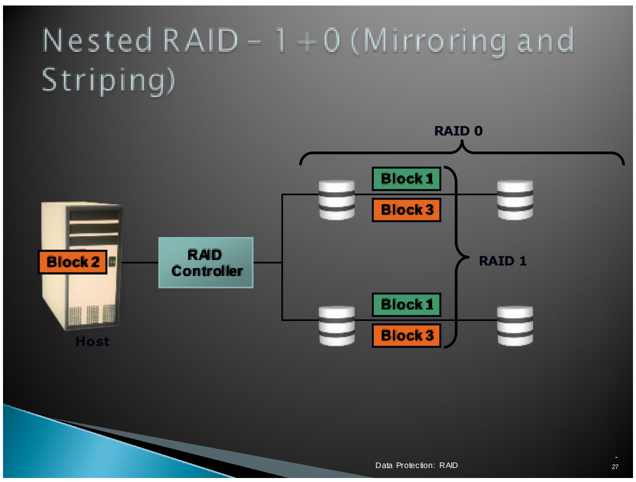


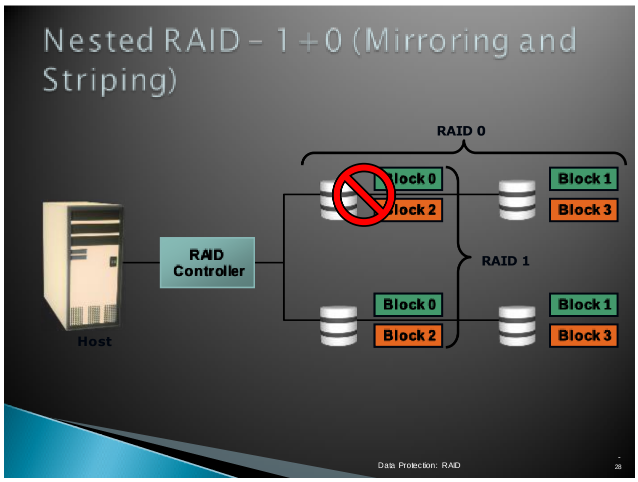
RAID 1 – Disk Mirroring




Nested RAID – 0+1 (Striping and Mirroring)
RAID 1
Block 0
Block 2
Block 1
Block 3
RAID 0
Block 0
RAID Controller
Host
Data Protection: RAID
25


Nested RAID – 0+1 (Striping and Mirroring)
RAID 1
Block 0
Block 2
Block 1
Block 3
Block 3
RAID 0
RAID Controller
Host
Data Protection: RAID
26

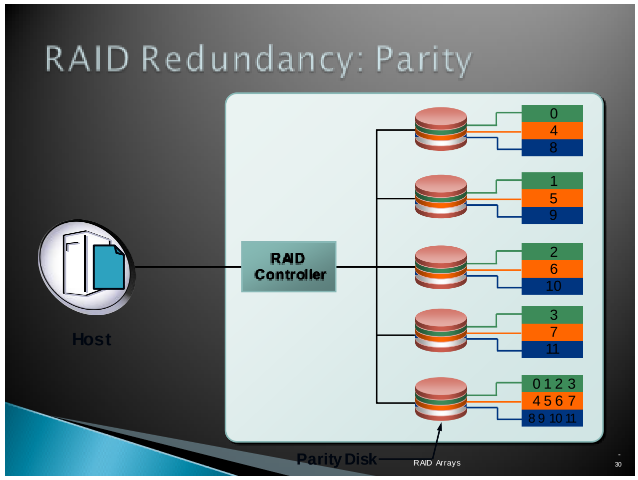

Nested RAID – 1+0 (Mirroring and Striping)
RAID 0
Block 1
Block 3
Block 1
Block 3
RAID 1
Block 2
RAID Controller
Host
Data Protection: RAID
27

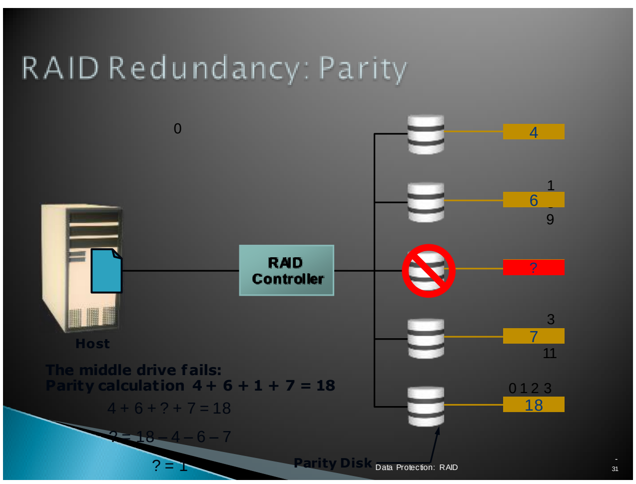

Nested RAID – 1+0 (Mirroring and Striping)
RAID 0
Block 1
Block 3
Block 0
Block 2
Block 1
Block 3
RAID 1
RAID Controller
Host
Data Protection: RAID
28

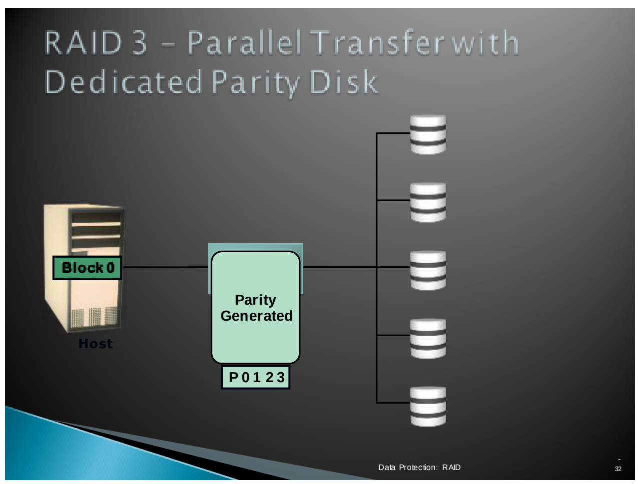
RAID 0+1 vs. RAID 1+0

- Benefits are identical under normal operations
- Rebuild operations are very different
  - RAID 1+0 uses a mirrored pair – only 1 disk is rebuilt if a disk fails
  - RAID 0+1 if a single drive fails, the entire stripe is faulted
    - RAID is 0+1 is a poorer solution and is less common

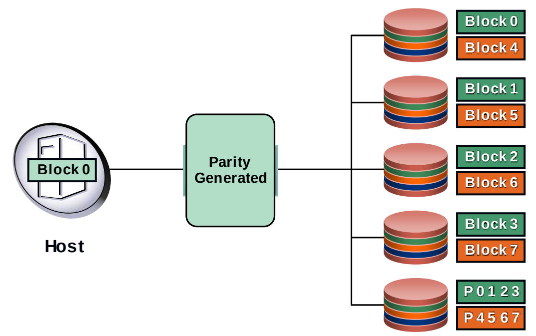
RAID Arrays 29



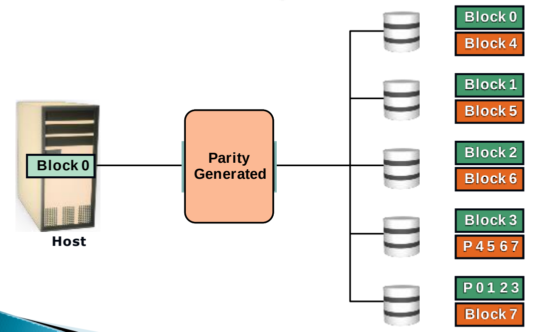




RAID 4 – Striping with Dedicated Parity Disk



RAID 5 – Independent Disks with Distributed Parity



Buffer-Replacement Policies (Cont.)

- **Pinned block** – memory block that is not allowed to be written back to disk.
- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- Most recently used (MRU) strategy – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer managers also support forced output of blocks for the purpose of recover

- To implement such specific policies a (good) database management system must usually implement its own buffer replacement policy (not relying in that of the operating system!)

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records*. A record is a sequence of fields.
- First approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

  This case is the easiest to implement.

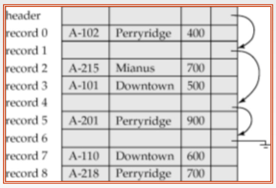
Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte *n* ∗ (*i* – 1), where *n* is the size of each record.
  - Record access is simple but records may cross blocks!
    - Modification: do not allow records to cross block boundaries
- Deletion of record *i*: alternatives:
  - move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

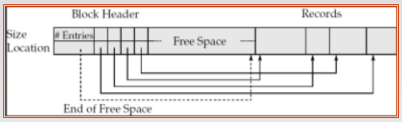
record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Free Lists

- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they "point" to the location of a record.
- More space efficient representation: reuse space for normal attributes of free records to store pointers. (No pointers stored in in-use records.)



Variable-Length Records: Slotted Page Structure



- Slotted page are usually the size of a block
- Header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- (Other) pointers should not point directly to record — instead they should point to the entry for the record in header.

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.
- If slotted pages are the size of a block, the issue of records spanning over more than one block is eliminated
- This limits the size of records in a database, which is usually the (default) case
  - There are special types for big records, that are treated differently (remember the **clobs** and **blobs** in Oracle?)

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O
- The choice of a proper organization of records in a file is important for the efficiency of real databases!

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	
A-888	North Town	800	

Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization.

customer_name	*account_number*
Hayes	A-102
Hayes	A-220
Hayes	A-503
Turner	A-305

customer_name	*customer_street*	*customer_city*
Hayes	Main	Brooklyn
Turner	Putnam	Stamford

Multitable Clustering File Organization (cont.)

Multitable clustering organization of *customer* and *depositor:*

Hayes	Main	Brooklyn
Hayes	A-102	
Hayes	A-220	
Hayes	A-503	
Turner	Putnam	Stamford
Turner	A-305	

- good for queries involving *depositor* ⋈ *customer*, and for queries involving one single customer and his accounts
- bad for queries involving only customer
  - but one can add pointer chains to link records of a particular relation
- results in variable size records

File System

- In sequential file organization, each relation is stored in a file
  - One may rely in the file system of the underlying operating system
- Multitable clustering may have significant gains in efficiency
  - But this may not be compatible with the file system of the operating system
- Several large scale database management systems do not rely directly on the underlying operating system
  - The relations are all stored in a single (multitable) file
  - The database management system manages the file by itself
  - This requires the implementation of an own file system inside the DBMS

Data Dictionary Storage

Data dictionary (also called system catalog) stores metadata; that is, data about data, such as

- Information about relations
  - names of relations
  - names and types of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices (next lecture)

Data Dictionary Storage (Cont.)

- Catalog structure
  - Relational representation on disk
  - specialized data structures designed for efficient access, in memory
- A possible catalog representation:

Relation_metadata = (*relation_name*, *number_of_attributes*, *storage_organization*, *location*)
Attribute_metadata = (*attribute_name*, *relation_name*, *domain_type*, *position*, *length*)
User_metadata = (*user_name*, *encrypted_password*, *group*)
Index_metadata = (*index_name*, *relation_name*, *index_type*, *index_attributes*)
View_metadata = (*view_name*, *definition*)

File Organization in Oracle

- Oracle has its own buffer management, with complex policies
- Oracle doesn't rely on the underlying operating system's file system
- A database in Oracle consists of **tablespaces**:
  - System tablespace: contains catalog meta-data
  - User data tablespaces
- The space in a tablespace is divided into **segments**:
  - Data segment
  - Index segment
  - Temporary segment (for sort operations)
  - Rollback segment (for processing transactions)
- Segments are divided into **extents**, each extent being a set of contiguous **database blocks**.
  - A database block need not be the same size of an operating system block, but is always a multiple

File Organization in Oracle (cont.)

- A standard table is organized in a heap (no sequence is imposed)
- Partitioning of tables is possible for optimization
  - Range partitioning (e.g by dates)
  - Hash partitioning
  - Composite partitioning
- Table data in Oracle can also be (multitable) clustered
  - One may tune the clusters to significantly improve the efficiency of query to frequently used joins.
- Hash file organization (to be studied in the sequel) is also possible for fetching the appropriate cluster
- A database can be tuned by an appropriate choice for the organization of data:
  - Choosing partitions
  - Appropriate choice of clusters
  - Hash or sequential
- Tuning makes the difference in big (real) databases!