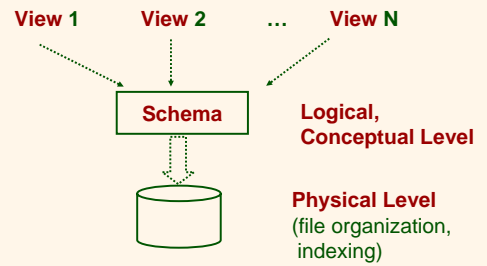




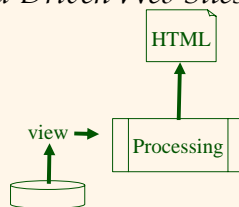
DBMS Physical Design: Disks and Files



The Three-level Architecture for Databases



Presentation Layer (4th Tier): Data-Driven Web Sites



- “Data driven web sites” also add an HTML “presentation” layer on top of what we’ve seen
- Or they use XML plus “style sheets” to get the same effect



Why study physical schema design?

- **Systems designers require knowledge of data organization techniques for efficient DBMS implementation**
 - > **system performance is key to success of the database application**
 - **Role of the Database Administrator (DBA)**

Database Physical Level

- Conceptual and logical levels for DB users
- Goal of DBMS is to simplify and facilitate access to data
- Performance is major factor in user satisfaction
 - > Response time for request must be short
- Performance depends on physical design parameters
 - > Data structures used and how efficiently implemented
 - > OS only provides basic services

System Performance: The AAA thumb rule

- Application
- Algorithm
- Architecture
 - > Not that different for Database applications
 - Architecture here is the physical level design
 - How files are stored, what indices
 - Algorithm here is the manner in which SQL operators are implemented
 - What is the algorithm used for a GROUPBY, Join etc.

DBMS: Physical System Structure

- Query parser and optimizer
- File manager
 - > Manages allocation of space on disk & data structures
 - > Buffer/Page management
- Recovery manager
 - > Ensures database remains consistent despite failures
- Concurrency controller
 - > Ensures correct concurrent access to data

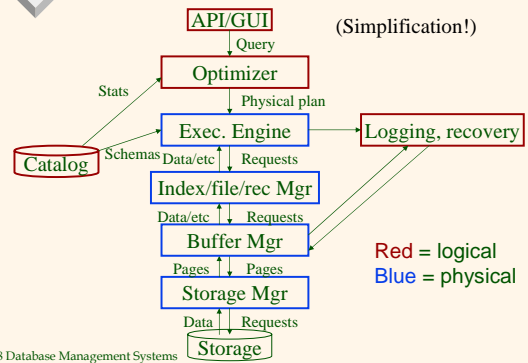
DBMS: Physical System Structure

- Query parser and optimizer
 - > Parse the SQL query, translate into intermediate form
 - Intermediate form: relational algebra!
 - > Rewrite query into an equivalent but more efficient form
- Query processing
 - > How to implement the relational operators
- File manager
 - > Manages allocation of space on disk & data structures
 - > Buffer/Page management
- Recovery manager
 - > Ensures database remains consistent despite failures
- Concurrency controller
 - > Ensures correct concurrent access to data

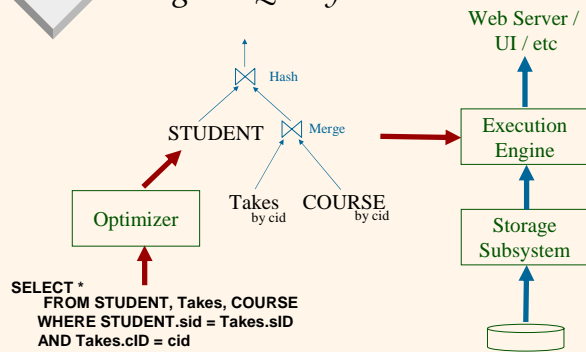
DBMS: Steps in Physical Layers

- Parse SQL program and generate intermediate code (no different from compiler) – intermediate code is rel.algebra expression!
- Optimize the query into a more efficient form
- Generate 'object' code (i.e., C) for relational operators
- Store data on disk using an efficient scheme
- Specify indices for relations
- Invoke file manager to fetch pages from disk
- Invoke concurrency controller for concurrent transactions
- Log the transactions to support recovery from failures

The Layers of the DBMS



Processing the Query



Data Organization

- Data structures part of physical organization
 - > **Data files:** store database itself
 - > **Data dictionary/catalog:** stores info about structures of database and info on tables in tablespace
 - > **Indices:** provide fast access to data
 - > **Statistical data:** for query optimization

Physical Storage: Memory Hierarchy

- **Primary Storage: cache & main memory**
 - Can be directly accessed by CPU
 - Currently used data
- **Secondary Storage: magnetic disks, optical disks, tapes**
 - Larger capacity, low cost, slow access
 - Cannot be directly processed by CPU
- **DB stores large amount, persist over time**
 - Data is stored in secondary storage
 - Contrast with run-time data structures
- **Time taken to fetch data depends on how data is organized on disk/file**

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.**
- **Main memory is volatile.**
 - We want data to be saved between runs. (Obviously!)
 - Situations that cause permanent loss of data occur less frequently in disks than primary memory
 - Disk storage is **non-volatile**

Disks

- **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.**
- **Unlike RAM, time to retrieve a disk page varies depending upon location on disk.**
 - Therefore, relative placement of pages on disk has major impact on DBMS performance!

Disk Storage

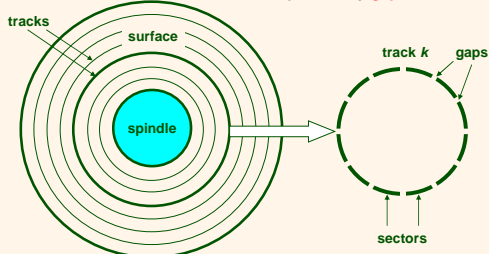
- **Capacity of disk: number of bytes it can store**
- **Disks made of magnetic material**
 - Encoding of bits into magnetic fields
 - **What about optical disks ?**
- **Disks assembled into disk packs**
- **Information stored on disk surface in concentric circles each having distinct diameter**
 - tracks

How do disks work ?

- **Recall from CS135**

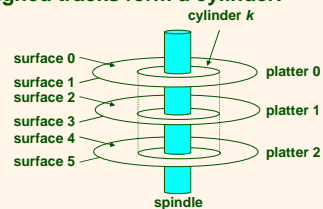
Disk Geometry

- Disks consist of **platters**, each with two **surfaces**.
- Each surface consists of concentric rings called **tracks**.
- Each track consists of **sectors** separated by **gaps**.



Disk Geometry (Multiple-Platter View)

- **Aligned tracks form a cylinder.**



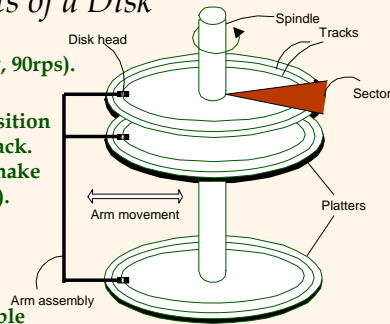
Components of a Disk

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 **sector size** (which is fixed).



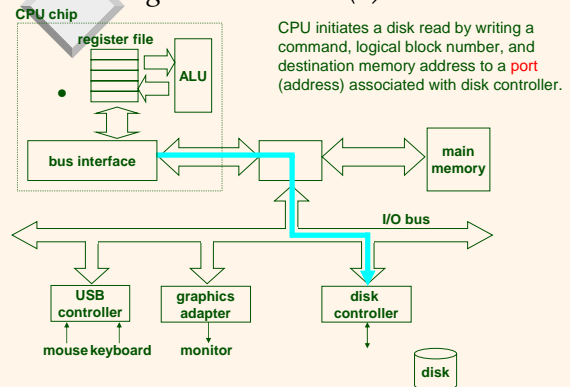
Disk Access Time

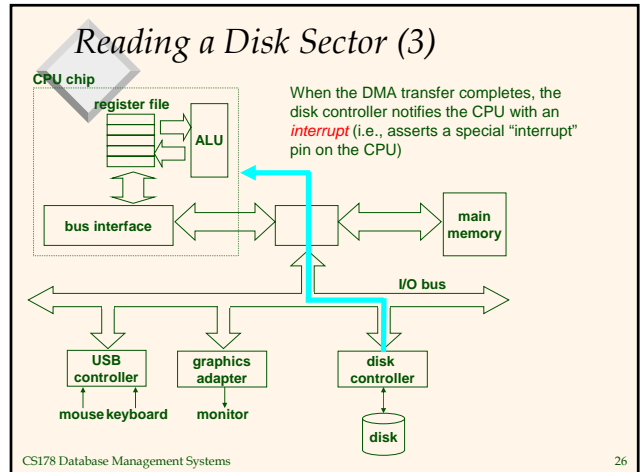
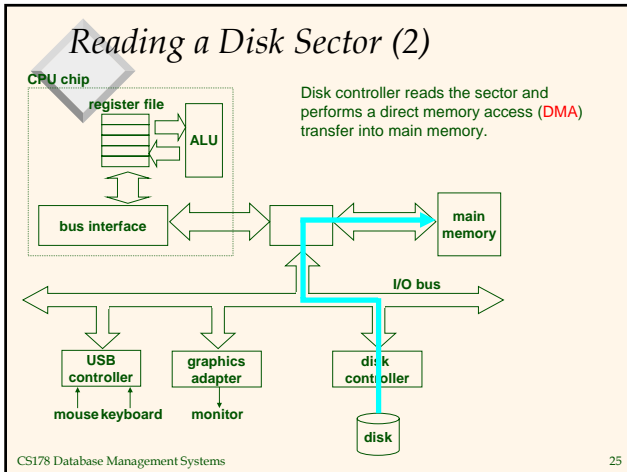
- **H/W address of disk block:** (surface #, track #, sector #)
- **Average time to access a target sector approximated by :**
 - $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$
- **Seek time ($T_{\text{avg seek}}$)**
 - Time to position heads over cylinder containing target sector.
 - Typical $T_{\text{avg seek}} = 9 \text{ ms}$
- **Rotational latency ($T_{\text{avg rotation}}$)**
 - Time waiting for first bit of target sector to pass under r/w head.
 - $T_{\text{avg rotation}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min}$
- **Transfer time ($T_{\text{avg transfer}}$)**
 - Time to read the bits in the target sector.
 - $T_{\text{avg transfer}} = 1/\text{RPM} \times 1/(\text{avg \# sectors}/\text{track}) \times 60 \text{ secs}/1 \text{ min}$

Disk Structure

- **For READ/WRITE operations**
 - H/W address of block and address of **buffer** is supplied to disk IO hardware via **disk controller**
 - Buffer is contiguous reserved area in main memory that holds block (page)
- **Actual H/W that reads blocks is disk head, part of disk drive**
- **Disk drives rotate disk pack**
- **Disk arm positions disk head over block read**
 - When block passes under disk head, data transferred to buffer

Reading a Disk Sector (1)





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.**
- **Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?**

CS178 Database Management Systems 27

Logical Disk Blocks

- **Modern disks present a simpler abstract view of the complex sector geometry:**
 - > The set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)
- **Mapping between logical blocks and actual (physical) sectors**
 - > Maintained by hardware/firmware device called disk controller.
 - > Converts requests for logical blocks into (surface, track, sector) triples.

CS178 Database Management Systems 28

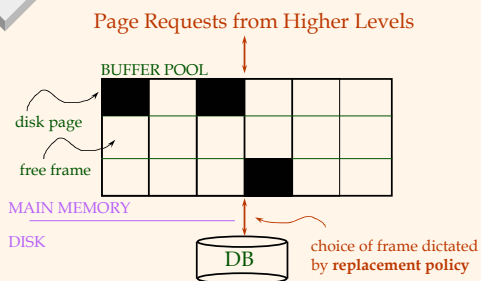
Arranging Pages on Disk

- **`Next' block concept:**
 - blocks on same track, followed by
 - blocks on same cylinder, followed by
 - blocks on adjacent cylinder
- **Blocks in a file should be arranged sequentially on disk (by `next'), to minimize seek and rotational delay.**
- **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.

Buffer Management in a DBMS



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

When a Page is Requested ...

- **If requested page is not in 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/Page Replacement Policy

- Main memory pages are full, need to 'replace' a page – throw it out of RAM and bring in new page
- Frame is chosen for replacement by a **replacement policy**:
 - Least-recently-used (LRU), Clock, MRU 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.

DBMS vs. OS File System

OS does disk space & buffer mgmt: 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 CC & recovery from failure),
 - adjust replacement policy, and pre-fetch pages based on access patterns in typical DB operations.

Buffer/Page Management in DBMS

- Replacement strategy
 - When no room left on buffer, a page must be removed from buffer
- Pinned blocks
 - For DB to recover from failures may need to restrict times when page is written to disk
- Forced output of blocks
 - For recovery and fault tolerance, block may have to be written even if space exists

Why tailor File Manager to DBMS

- Since DBMS executes relational queries, patterns known in advance
- Example: consider computation of query $\text{loan} \otimes \text{customer}$
 - Use simple iterative join algorithm
 - Schemas:
 - $\text{loan}(\text{custID}, \text{loan-number}, \text{branch-name}, \text{amount})$
 - $\text{Customer}(\text{custID}, \text{name}, \text{street}, \text{city})$

Example: Join Algorithm

```
For each tuple b in loan do
    for each tuple c in customer do
        if b[custID] = c[custID]
            then begin
                create the join tuple x
            endif
```

Page replacement for Join Algorithm

- Once tuple of *loan* is processed it is never referenced again
 - > Once a block of *loan* is processed it can be written back even it is most recently used
- Once a *customer* block is used it will not be referenced again till all other blocks of *customer* are referenced
 - > Least recently used block of *customer* will be next referenced!
- What replacement strategy to use ?

LRU vs MRU in Example

- LRU does exactly opposite of the access pattern
 - > Using the default OS LRU leads to disaster in terms of performance
- MRU is a better option for this access pattern
- In general: Tailoring the OS utilities to the DBMS leads to better design & performance
 - > Most commercial DBMS systems (Oracle, M-soft) have their specialized file management s/w

Next: File and Data Organization

- How is data stored on disk?
 - > Records
 - > file of records
- how to organize the files to enable fast processing of queries
 - > how to measure speed - computation or I/O time ?
- Study data organization techniques that lead to more efficient processing of queries

File and Record Organizations

- **DB applications typically need small portion of database**
 - > **when specific data needed:**
 - located on disk
 - copied into main memory
 - rewritten into disk if data changed
- **data stored on disk is organized as file of records**
 - > **File is a sequence of records**
 - **Records mapped to disk blocks**

Files 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 containing a collection of records. Must support:
 - > insert/delete/modify record
 - > read a particular record (specified using *rid: record id*)
 - > scan all records (possibly with some conditions on the records to be retrieved)

Mapping Relations to Files

- **Most DBMS store each relation in separate file**
 - > records correspond to rows
 - > record fields correspond to columns
 - > joins require accessing multiple files

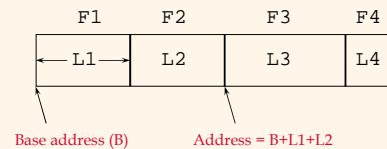
Organization of Records

- **Record is collection of related information**
 - > each value is one or more bytes, corresponds to a particular **field** of record
 - > each **field specifies some attribute**
 - > collection of field definitions and their types constitutes **record type** or format
 - data type associated with each field
 - > blocks are fixed size, but record sizes vary
- **fixed length records or variable length**
 - > Store in manner which facilitates fast access

Record Types

- **Fixed length vs Variable length records**
 - > fixed is easier to implement
 - > fixed wastes space when block size not multiple of record size
- **spanned vs unspanned**
 - > when parts of a record can be placed onto a block, need pointers to next block where remainder of record is placed

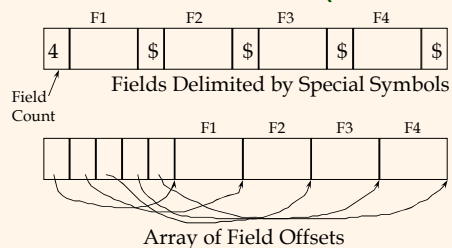
Record Formats: Fixed Length



- 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.

File Management

- Support search, scan, and insert/delete
- When records deleted or inserted, need to move records to occupy space or mark empty space
- maintain file header
 - > point to next record that is deleted
 - > first record point to next empty record etc.
 - > can have dangling pointer problem on delete
 - **pinned records**: avoid moving or deleting records that are pointed to by other records

Link between file organization and DBMS efficiency ?

File Organizations

- **File organization determines how records are physically placed on disk**
 - > heap file: no particular order
 - > Sorted file
 - > indexed file
 - hash index
 - tree indices
- **Efficiency of file organization typically measured in terms of number of disk accesses to fetch data**
 - > Why ?

Evaluation of File Organizations

- **Time always measured in # disk accesses**
- **Access time or lookup time**
 - > time to find particular data item
- **Insertion time**
 - > time to insert new record
 - time to find correct location and time to insert
- **Deletion time**
- **Modification time**
- **Space overhead**
 - > additional space occupied by index structure

Evaluation of File Organizations

- **Relation of size n records – n rows/tuples**
- **disk block size b bytes – page size**
- **record size r bytes**
 - > average size
- **“blocking factor” p , number of records/block**
 - > $p = b/r$
- **number of disk blocks to store relation**
 - > $\lceil n/p \rceil$

Example

- File of 1,000,000 records
- record size 200 bytes
- blocks are 4096 bytes
 - > $n = 1,000,000$
 - > $r = 200$
 - > $b = 4096$
 - > Blocking factor, $p = b/r = 4096/200 = 20$
 - > file size = $N = n/p = 1,000,000/20 = 50,000$ blocks

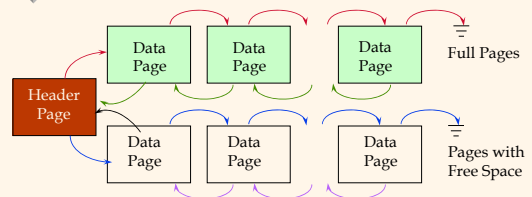
Evaluation of File Organizations

- Baseline we use Heap File
 - > The do nothing approach!
- Derive performance for each type of file organization
 - > note that query type plays a large role in determining efficiency

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.

Evaluation of Heap Files

- **Lookup time:** on average retrieve $\frac{1}{2}(n/p)$ I/Os
 - > worst case = $n/p = N$
- **insertion time:** retrieve last record on heap, if no empty space then start new block
 - > 2 disk I/O
- **deletion:** find record and then delete
 - > $\frac{1}{2}(n/p)+1$ average, $n/p+1$ worst case
- **modification:** same as deletion

Example

- **File of 1,000,000 records**
- **record size 200 bytes**
- **blocks are 4096 bytes**
 - > $n = 1,000,000$
 - > $r = 200$
 - > $b = 4096$
 - > **Blocking factor, $p = b/r = 4096/200 = 20$**
 - > **file size = $N = n/p = 1,000,000/20 = 50,000$ blocks**

Heap File Example

- **Successful lookup ?**
- **Insertion time ?**
- **Deletion time ?**
- **Modification ?**

Heap File: Example

- **Successful lookup: average $\frac{1}{2}(n/p) = 25,000$**
 - > worst case is $n/p = 50,000$ disk accesses
 - > At 10ms disk access time, this is 500 seconds ~ 8 minutes!
- **insertion = 2**
- **deletion = $\frac{1}{2}(n/p)+1 = 25,001$**
 - > worst case = 50,001
- **header page of pointers can get large**
- **Heap file summary: not a smart solution!**

Lesson 1: better organize the records on the file

- **Heap file will not cut it!**
- **Need to organize physical records on the file in some “smart” manner**
 - Sorted file
 - Hash file

Lesson 2: do we really need to access the entire file to answer a query ?

- **Many queries reference small portion records**
 - **system should be able to locate these without having to search all records**
 - **Without having to search through the physical file of records ?**

Lesson 2: Indexing

- **placing additional structure on files helps search efficiently for desired records**
 - Create an index structure on the actual data file

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, e.g.,**
 - 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.
- **Improve efficiency of operations by defining an index to the relation**

Sequential Files and Indexing

- **File can be sorted**
 - sequential file
 - can use binary search
- **When file gets large, create indices**
 - pointer to data
 - if number of pointers is large, then have pointers to set of pointers
 - can have multiple levels of indexing
 - **What is a multi-level index that you have used?**

An old, but common, indexing scheme...

- **Library!**

Library Catalogs

- **Data analogous to Books**
 - each book has unique library key(QA76.6 222)
 - books sorted on each shelf
 - shelves on each floor arranged by library key
- **search for book using index cards**
 - index cards has book title, key, and location of book (shelf/floor) in library
 - index cards stored (sorted) in “filing cabinet”
 - each shelf in cabinet indexed alphabetically!
 - **What if cabinets have to stored in multiple rooms?**

System Catalogs

- **For each index:**
 - structure (e.g., B+ tree) and search key fields
- **For each relation:**
 - name, file name, file structure (e.g., Heap file)
 - attribute name and type, for each attribute
 - index name, for each index
 - integrity constraints
- **For each view:**
 - view name and definition
- **Plus statistics, authorization, buffer pool size, etc.**

* *Catalogs are themselves stored as relations!*

Attr_Cat(attr_name, rel_name, type, position)

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Summary


- **Disks provide cheap, non-volatile storage.**
 - Random access, but cost depends on location of page on disk
- **Buffer/Page manager brings pages into RAM**
 - specialized page manager for DBMS
- **File keeps track of pages in a file: file=relation**
 - collection of records organized in some manner
- **Indexes support efficient retrieval of records based on the values in some fields.**
- **Next: efficient file organization and index structures**
 - Sorted files
 - hashing
 - B-trees
 - external sorting

CS 178: Term Project

- **Three phase project**
 - **Phase 1: "paper design"**
 - Individual, due March 1
 - Put down all the data you need, understand the workflow process, schema design
 - **Phase 2: Implement one application**
 - Two person teams, Due early April
 - Fine tune the schema, populate tables, implement the queries, implement the correct workflow
 - Bells & whistles can be left for Phase 3
 - Clear set of requirements will be specified
 - **Phase 3: Integrate both applications to produce one final "product"**
 - Two person teams, but each from different application in Phase 2
 - Integrate the two modules, implement the bells and whistles
 - Report generation, GUI, etc.

Term Project....

- **Teams and teamwork**
 - **Required – not an option**
 - Failure to work with team partner is automatic zero (for the project)
 - No exceptions/whining/excuses
 - **Specific lab dates will be announced for project discussions/teamwork – failure to attend these sessions will lead to a zero on the project**
 - **Minimum meeting time for teams: lab section**
 - **Each team member has specific "role"**
 - You have to specify the role in your Phase 2 report/demo



Term Project: Phase 3

- **Phase 3 is integration**
 - **Take the two pieces and integrate**
 - Real world: you will take pieces written by someone else
 - CS178 world: you will take pieces written by yourself and by your team partner and integrate
 - **Integration is NOT re-design**
 - Re-designed implementation will get you zero points on Phase 3!
- **You know integration is needed even if you do not know with which team/person**
 - **Keep this in mind during design of Phase 2!!!!**