CSC553 Advanced Database Concepts

Tanu Malik

School of Computing

DePaul University

Syllabus and Course Admininstration

- https://dice.cs.depaul.edu

What is a database?

What is a database?

- A *large* collection of *related* data
- A database is:
  - Persistent: stored on a stable medium
  - Shared: multiple (simultaneous) users
  - Interrelated: forms a bigger picture
- Models real world information

Database Management System

- DBMS is the software component that allows creating/maintaining/controlling access to a database
- Why not store data in a flat file?
  - E.g., .txt or .xls
  - Least amount of work!

Data retrieval

- Query = declarative data retrieval
  - *describes **what** data to retrieve, not **how** to retrieve it*
  - Example*: Give me the students with GPA > 3.5* VS
  - *Scan the student file and retrieve the records with GPA > 3.5*
- Why?
  - Easier to write
  - More efficient to execute

What is a data model?

A Data Model

- A notation for describing data or information.
- Consists of 4 parts:
  - Structure of the data
    - Data structures and relationships
  - Operations on the data
    - programming operations
  - Constraints on the data
    - Describe limitations on data
  - Persistence of data
    - Data is stored permanently on disk

Types of Data Models

- Based on files
  - Filesystem
- Based on tables
  - Relational data model
- Based on trees and graphs
  - Semistructured data model

Managing Data in Files

- How to find a particular record?
  - Find all students who joined later than 2013
  - Find all students who live in Chicago
- What if two threads try to write to the same file at the same time?
- How to ensure that the studentid remains same in all files?
- What is data is overwritten?

DBMS Evolution

- 1960s: Codasyl (network), IMS (hierarchical)
- 1970s: Relational Model (System R, Ingres)
- 1980: SQL as a language for RDBMS
- 1990: Object-Relational Model (disk-based databases)
- 2000’s: NoSQL
- 2010: NewSQL (Main-memory databases)

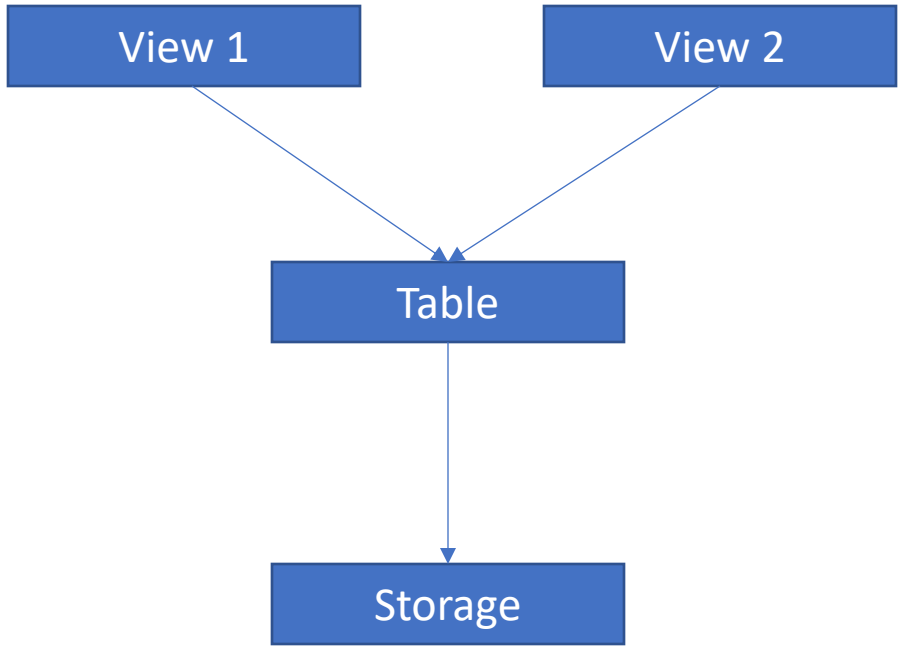
Relational Model

- Very popular
- Provides program-data independence
  - Create a 3 level architecture

Program-Data Independence

- View-Level
  - What data is exposed to users, what are they allowed to see
- Logical Level
  - Definition of tables, attributes, constraints, etc
- Physical Level
  - Data stored in files, location of files, indexed, storage layout

3-level architecture



Program-Data Independence

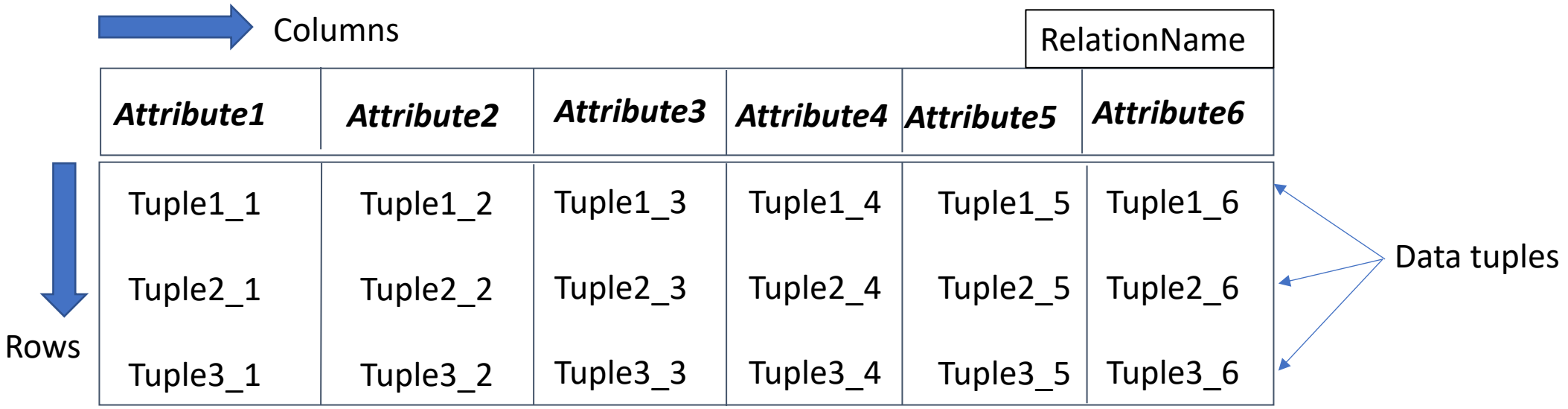
- Logical Data Independence
  - Modify table definitions without changing view used in an application
    - Add/drop/rename attributes from a table
    - Rename a table
- Physical Data Independence
  - Modify the structure and organizaton of physical storage with changing logical schema
    - Change the type of index
    - Compress a table when it is stored on disk
    - Move a table to another disk/machine

Relational Model

A relational model stores data in *relations*.

Relations are also known as tables.

Definition: A RELATION is A 2-Dimensional DATA STRUCTURE



RelationName, Attribute Names, Data tuples

A RELATION is A 2-Dimensional DATA STRUCTURE

Student

SID	*Firstname*	*Lastname*	*City*	*Started*	*Age*
192-83-332	Johnson	Epel	Palo Alto	2010	21
019-28-553	Smith	Shaw	Chicago	2010	21
192-83-290	Chan	Gloria	Palo Alto	2011	22
321-12-312	Marcus	Brenningan	Naperville	2013	23
019-28-321	Deepa	Patel	Chicago	2015	25

Rows

Properties: Attributes and Tuples

- Columns of a relation are named by attributes
  - Appear as the first row at the top.
  - Attributes do not store units.
- Rows of a relation (except first row) are called tuples
  - The first row is an attribute row and is typically not part of tuples.

Schema of a Relation

- Schema: the (i) name of the relation and (ii) the set of attributes for a relation is called the schema for that relation.
- Provides a concise representation of the relation without the data.

- Example:

Schema = **Student**(Sid, Firstname, Lastname, City, Started, Age)

Domain of An Attribute

- Each attribute is associated with a domain or an elementary type.
- Domains often implicit in schema
- Example:

Student(SID:integer; FirstName: string; LastName: string; City: string; Started: integer; Age: integer)

The Following is not an Instance Of The Student Relation

SID	*Firstname*	*Lastname*	*City*	*Started*	*Age*
192-83-332	Johnson	Epel	Palo Alto	2010	21
abcdef	Smith	Shaw	12345	2010	21
192-83-290	Chan	Gloria	Palo Alto	Abc_34	22
321-12-312	Marcus	Brenningan	Naperville	2013	23
ghijk	Deepa	Patel	Chicago	2015	25

TWO Domain Requirements: #1

1. Elementary type of domain always includes the special NULL, which is used if a value of a domain is unknown or missing.

Student

SID	*Firstname*	*Lastname*	*City*	*Started*	*Age*
192-83-332	Johnson	Epel	Palo Alto	2010	21
019-28-553	Smith	Shaw	NULL	2010	21
192-83-290	Chan	Gloria	Palo Alto	2011	NULL
321-12-312	Marcus	Brenningan	Naperville	2013	23
019-28-321	Deepa	Patel	Chicago	2015	25

TWO DOMAIN REQUIREMENTS: #2

2. Domain can only be atomic i.e., it can only be an elementary type such as String, Integer, Char, Float, Date.

$\Rightarrow$Domain cannot be a record structure, set, array, list, or any data structure whose values can be broken to smaller components.

Example

- Suppose we must represent an additional attribute “Contact Number” in Student table.
- Further, a student may have more than one contact numbers.
- What domain to assign to Contact Number attribute?

- Can we say:

Student (SID:Integer, FistName: string; LastName; string; City: String; Started:Integer; Age: Integer; Contact Number: List of Integers);

AN INVALID RELATION

- **Student**(SID:integer; FirstName: string; LastName: string; City: string; Started: integer; Age: integer; ContactNumber: List of Integers)

Student

SID	*Firstname*	*Lastname*	*City*	*Started*	*Age*	*ContactNumber*
192-83-332	Johnson	Epel	Palo Alto	2010	21	[773-712-3456 312-721-4561]
019-28-553	Smith	Shaw	NULL	2010	21	[312-362-1121, NULL]
192-83-290	Chan	Gloria	Palo Alto	2011	NULL	[312-362-3546, 312-462-3546]
321-12-312	Marcus	Brenningan	Naperville	2013	23	[773-752-3561, NULL]
019-28-321	Deepa	Patel	Chicago	2015	25	[773-386-5634, NULL

Representation 1: ADD COLUMN

- A valid relational representation
- **Student**(SID:integer; FirstName: string; LastName: string; City: string; Started: integer; Age: integer; CN1: Integer;

Student

SID	*Firstname*	*Lastname*	*City*	*Started*	*Age*	*CN1*	*CN2*
192-83-332	Johnson	Epel	Palo Alto	2010	21	773-712-3456	312-721-4561
019-28-553	Smith	Shaw	NULL	2010	21	312-362-1121	NULL
192-83-290	Chan	Gloria	Palo Alto	2011	NULL	312-362-3546	312-462-3546
321-12-312	Marcus	Brenningan	Naperville	2013	23	773-752-3561	NULL
019-28-321	Deepa	Patel	Chicago	2015	25	773-386-5634	NULL

Representation II: Add ROW

- Another valid relational representation
- **Student**(SID:integer; FirstName: string; LastName: string; City: string; Started: integer; Age: integer; CN: Int

Student

SID	*Firstname*	*Lastname*	*City*	*Started*	*Age*	*CN*
192-83-332	Johnson	Epel	Palo Alto	2010	21	773-712-3456
192-83-332	Johnson	Epel	Palo Alto	2010	21	312-721-4561
019-28-553	Smith	Shaw	NULL	2010	21	312-362-1121
192-83-290	Chan	Gloria	Palo Alto	2011	NULL	312-362-3546
192-83-290	Chan	Gloria	Palo Alto	2011	NULL	312-462-3546
321-12-312	Marcus	Brenningan	Naperville	2013	23	773-752-3561
019-28-321	Deepa	Patel	Chicago	2015	25	773-386-5634

Tuple Requirement: Relations are Sets

- Relations are *set* of tuples/rows and not a *list* of tuples/rows.

$\Rightarrow$The order in which tuples are listed in a relation does not matter

$\Rightarrow$The order in which columns are listed in a relation does not matter

SID	Firstname	Lastname	City	Started	Age
192-83-332	Johnson	Epel	Palo Alto	2010	21
019-28-553	Smith	Shaw	NULL	2010	21
192-83-290	Chan	Gloria	Palo Alto	2011	NULL
321-12-312	Marcus	Brenningan	Naperville	2013	23
019-28-321	Deepa	Patel	Chicago	2015	25

Different tuple orders

SID	Firstname	Lastname	City	Started	Age
192-83-332	Johnson	Epel	Palo Alto	2010	21
019-28-553	Smith	Shaw	NULL	2010	21
321-12-312	Marcus	Brenningan	Naperville	2013	23
192-83-290	Chan	Gloria	Palo Alto	2011	NULL
019-28-321	Deepa	Patel	Chicago	2015	25

SID	Firstname	Lastname	City	Started	Age
192-83-332	Johnson	Epel	Palo Alto	2010	21
019-28-553	Smith	Shaw	NULL	2010	21
192-83-290	Chan	Gloria	Palo Alto	2011	NULL
321-12-312	Marcus	Brenningan	Naperville	2013	23
019-28-321	Deepa	Patel	Chicago	2015	25

Different column orders

SID	Firstname	Lastname	Started	City	Age
192-83-332	Johnson	Epel	2010	Palo Alto	21
019-28-553	Smith	Shaw	2010	NULL	21
321-12-312	Marcus	Brenningan	2013	Naperville	23
192-83-290	Chan	Gloria	2011	Palo Alto	NULL
019-28-321	Deepa	Patel	2015	Chicago	25

Relation Constraints
Primary Key (PK)

- A set of attributes forms a *primary key* which has unique values in that set of attributes.
- A set of attributes forms a *primary key* for a relation if we do not allow two tuples in a relation instance to have the same values in ALL the attributes of the key.
- Student(SID, Firstname, Lastname, City, Started, Age)

University Database schema

Course	
CID	**int**
CourseName	varchar(40)
Department	varchar(4)
CourseNr	char(3)

Enrolled	
StudentID	int
CourseID	int
Quarter	varchar(6)
Year	number(4)

Student	
LastName	varchar(40)
FirstName	varchar(40)
SID	**int**
SSN	int
Career	varchar(4)
Program	varchar(10)
City	varchar(40)
Started	int

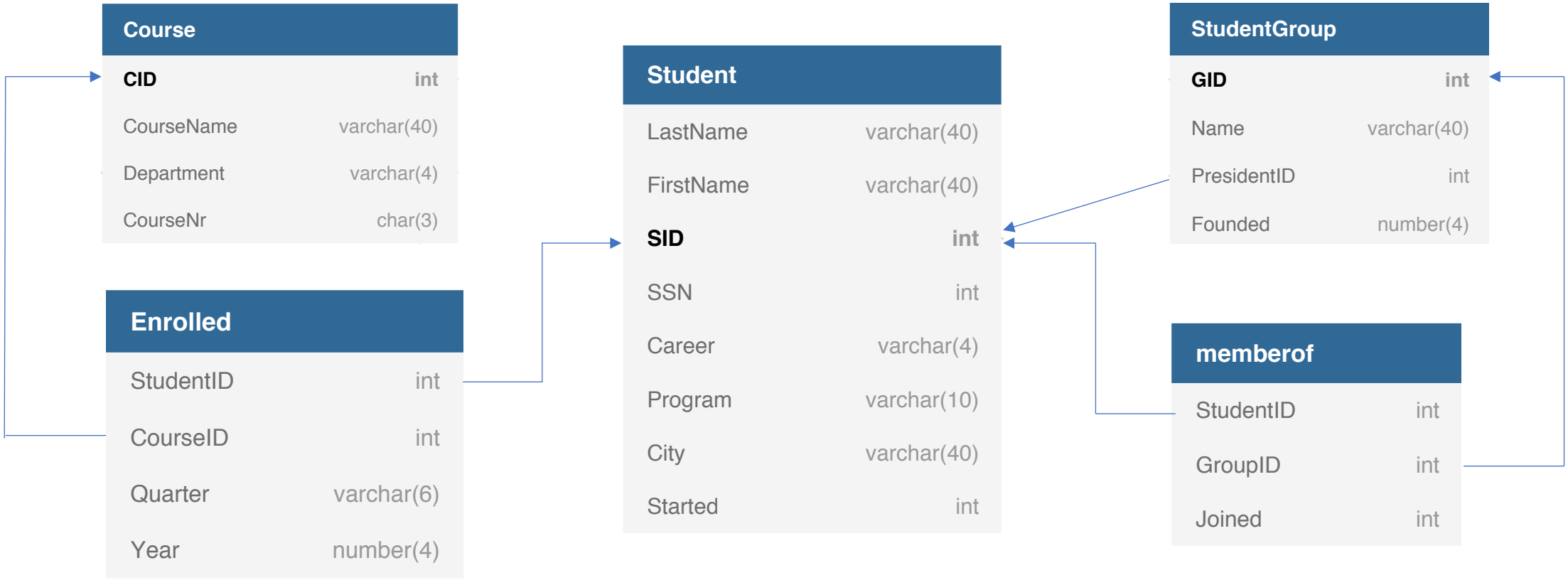
StudentGroup	
GID	**int**
Name	varchar(40)
PresidentID	int
Founded	number(4)

memberof	
StudentID	int
GroupID	int
Joined	int

Foreign key constraint

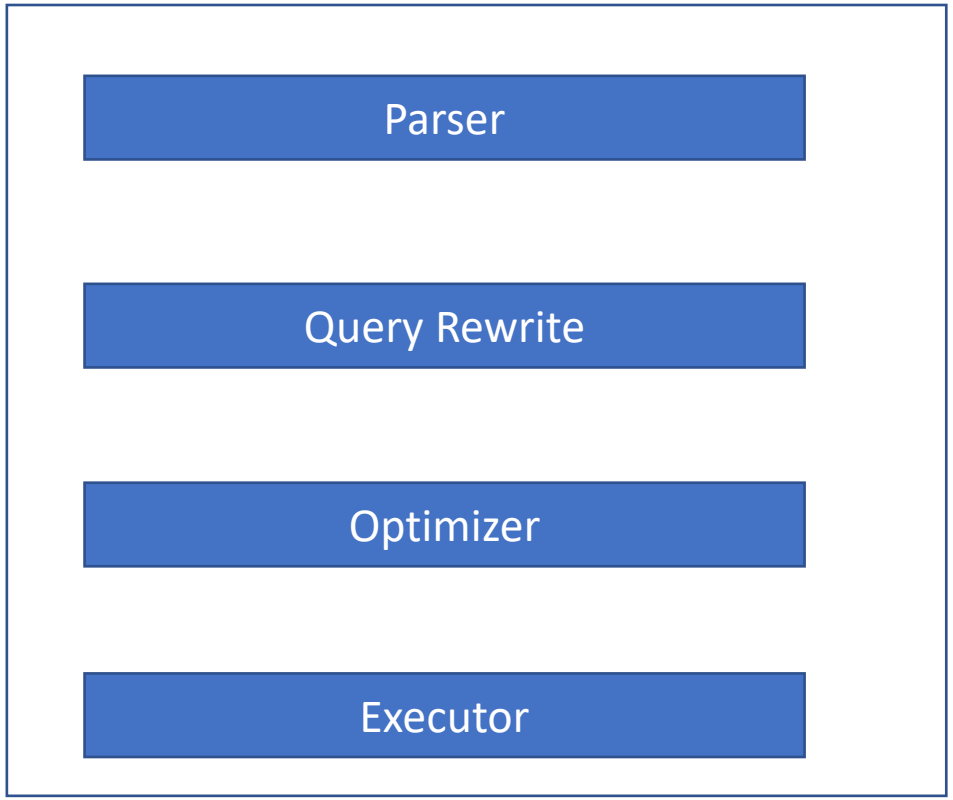
- Is a constraint from attribute(s) A of relation R1 to the primary key of B of relation R2, the value of A for each tuple in R1 must also be the value of B of some tuple in R2
- Attribute A is the foreign key from R1 referencing R2
- R1 = referencing relation
- R2 = referenced relation

University DB schema

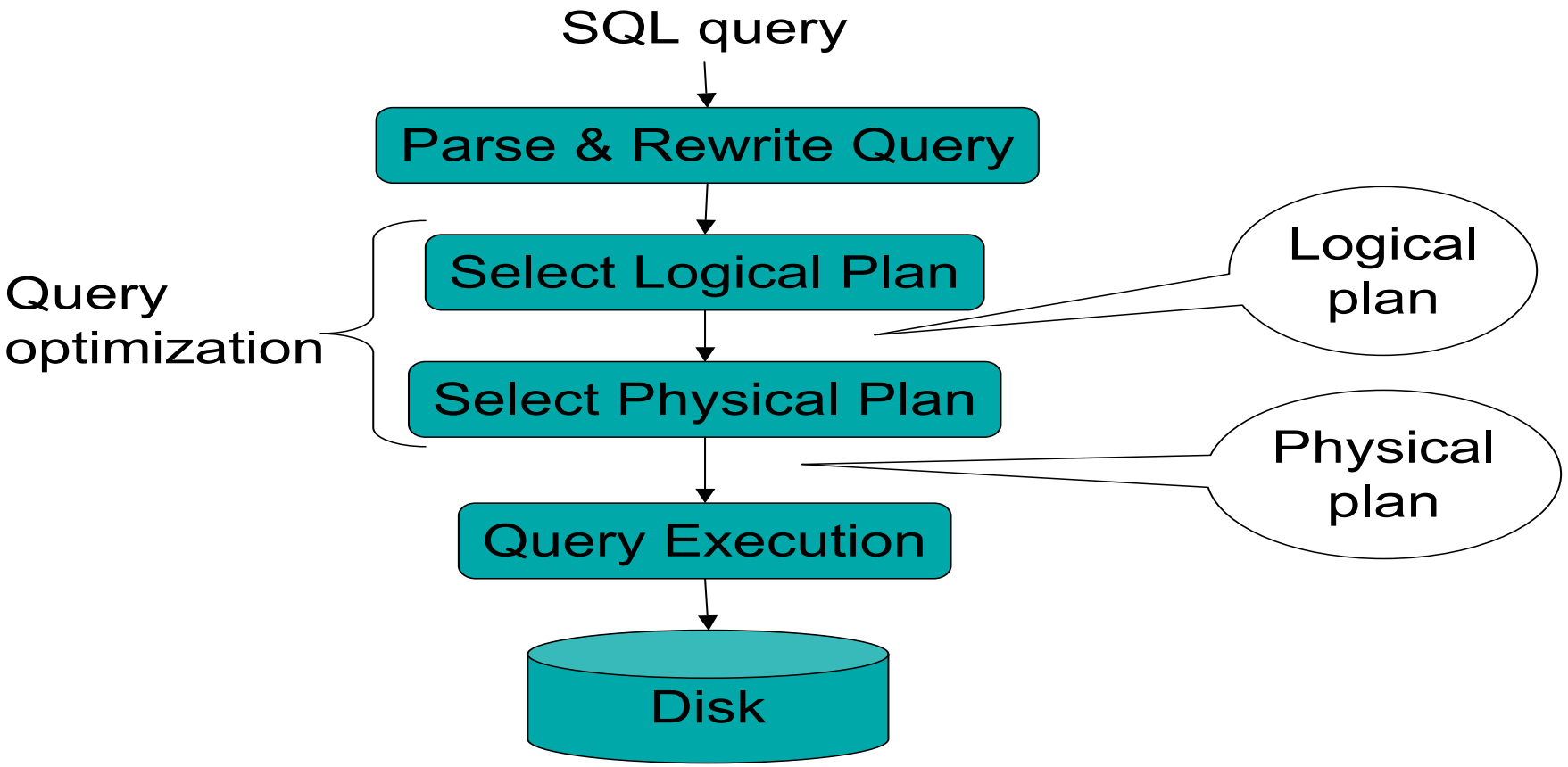


Database Internals

DBMS Architecture: Query Processor



Query Evaluation Steps



Example Database Schema

- Supplier(sno, sname, scity, sstate)
- Part(pno, pname, psize, pcolor)
- Supply(sno, pno, price)

- View

```
CREATE VIEW NearbySupp As
SELECT sno, sname
FROM Supplier
Where scity = ‘Chicago’ and sstate = ‘IL’
```

Example Query

- Find the names of all suppliers in Seattle who supply Part 2

```
SELECT sname FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Steps in Query Evaluation

- **Step 0: admission control**
  - User connects to the db with username, password
  - User sends query in text format
- **Step 1: Query parsing**
  - Parses query into an internal format
  - Performs various checks using catalog
- **Step 2: Query rewrite**
  - View rewriting, flattening, etc.

Rewritten Version of Our Query

- Original Query:

```
SELECT sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

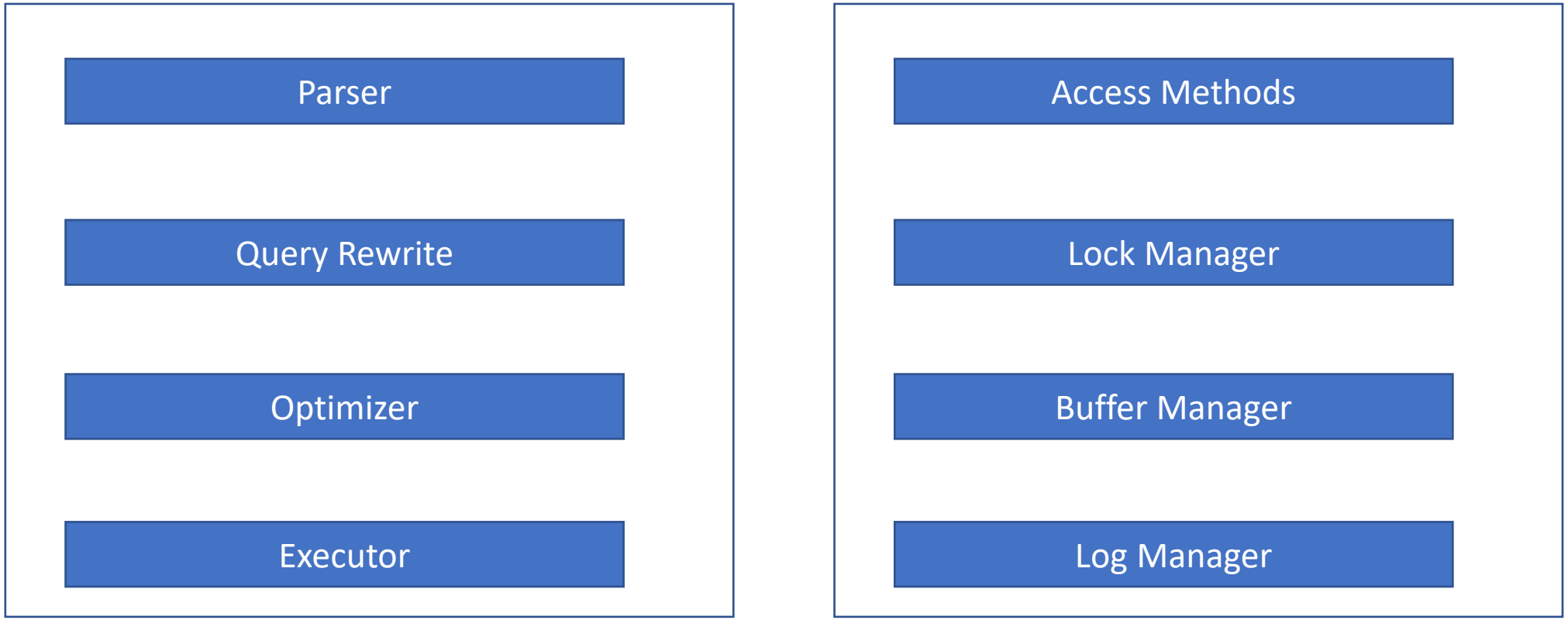
- Rewritten Query:

```
SELECT S.sname
FROM Supplier S, Supplies U
Where S.scity = 'Chicago' AND S.sstate = 'IL'
AND s.sno = U.sno AND U.sno = 2
```

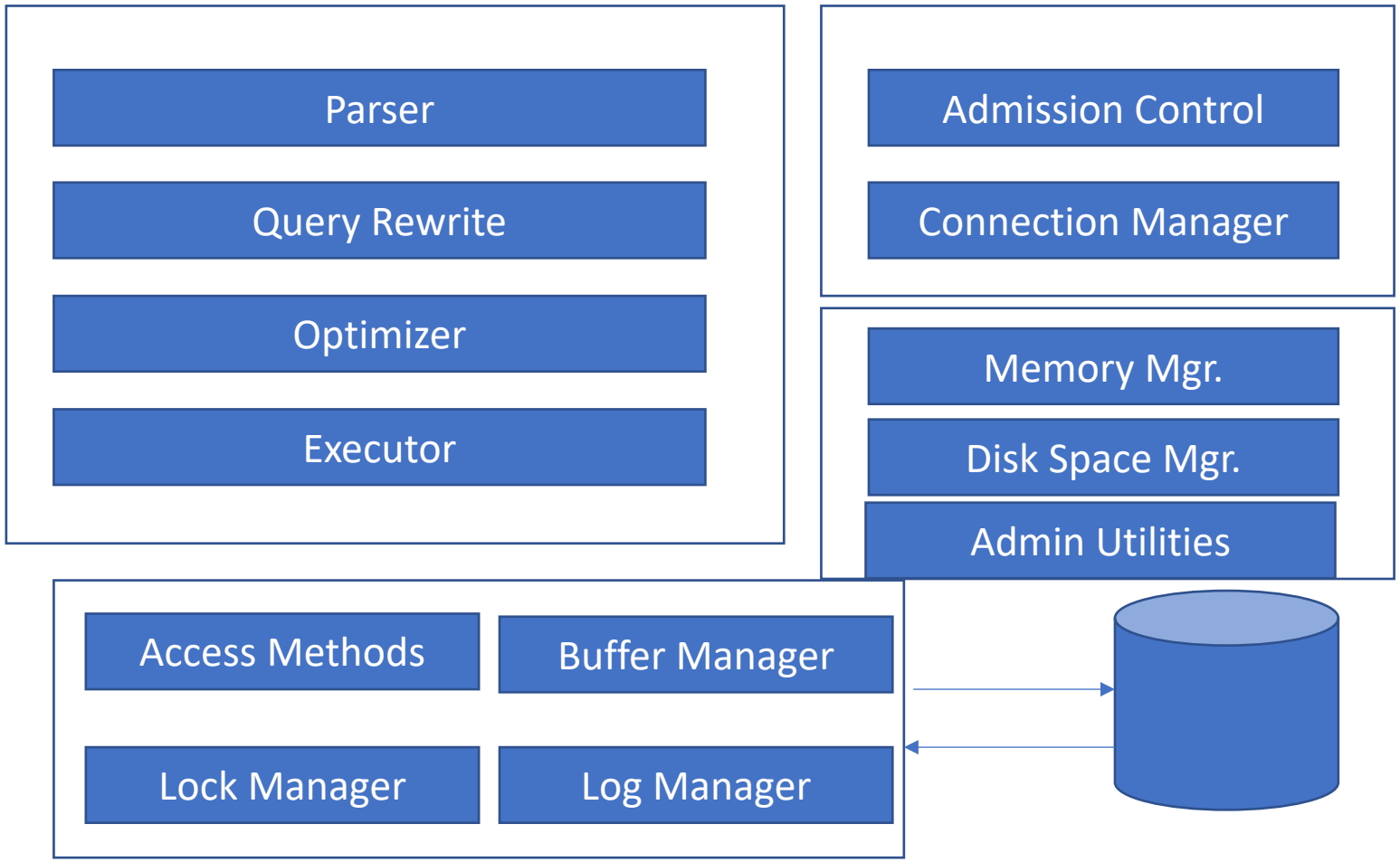
Continue with Query Evaluation

- **Step 3: Query optimization**
  - Find an efficient query plan for executing the query
  - We will spend a a lot of time on this topic
- A **query plan** is
  - **Logical query plan**: an extended relational algebra tree
  - **Physical query plan**: with additional annotations at each node
    - Access method to use for each relation
    - Implementation to use for each relational operator

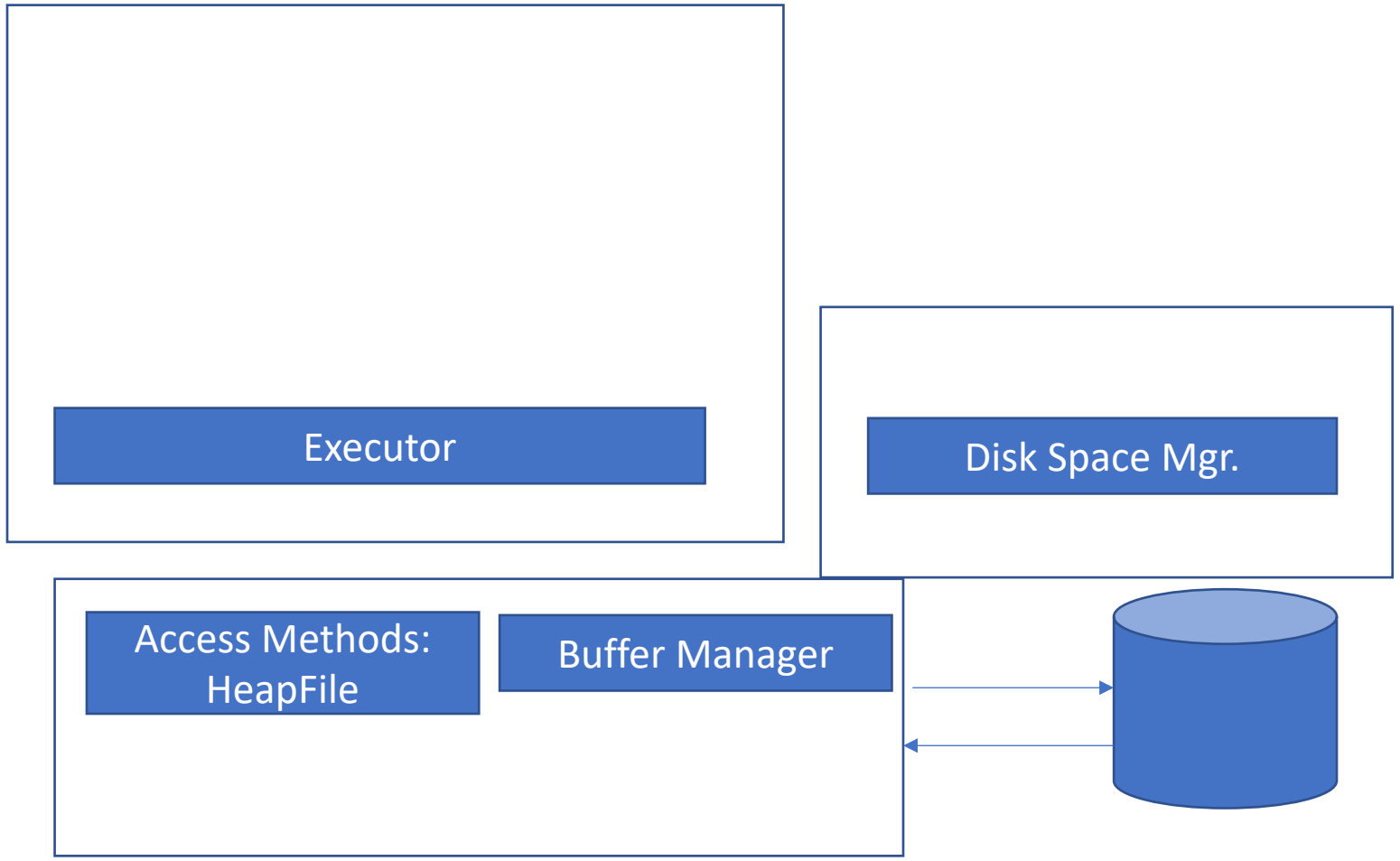
DBMS Architecture: Storage Manager



DBMS Architecture: Process Manager

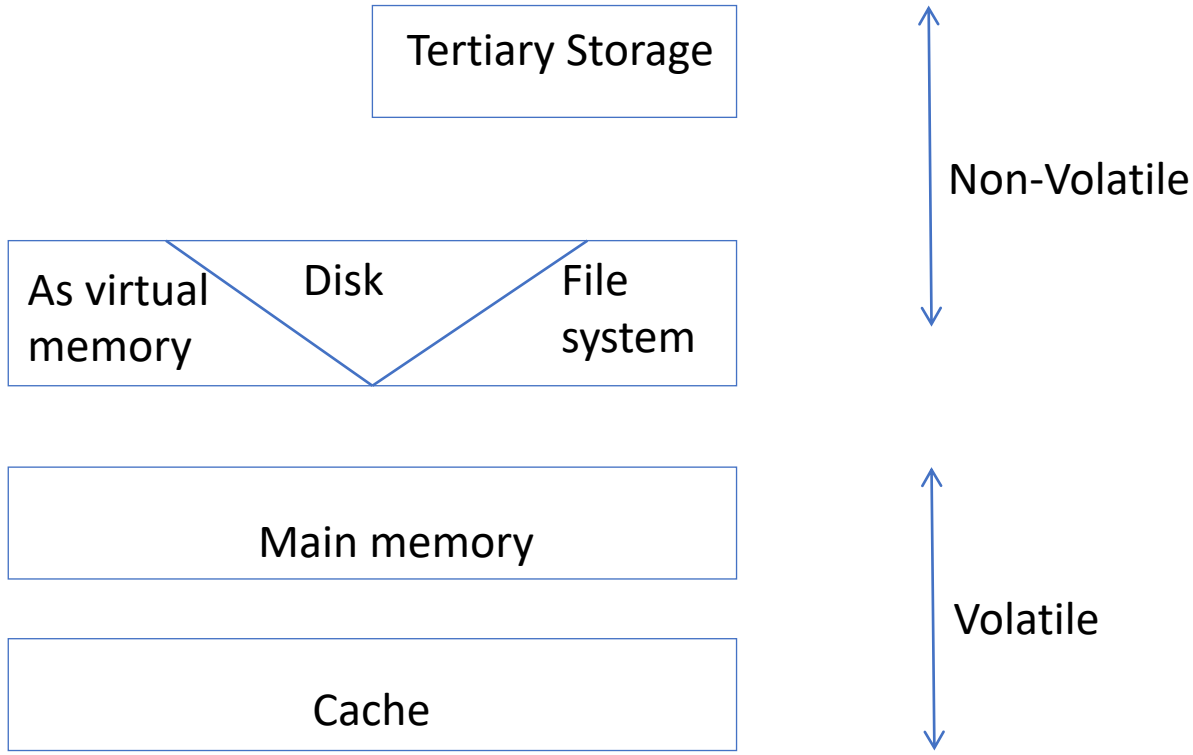


Storage Manager



- Process data with a simple query.
- Access methods: Organize data to support fast access to desired subsets of records.
- Buffer manager: Caches data in memory. Reads/writes data to/from disk as needed
- Disk-space manager: Allocates space on disk for files/access methods

The Memory Hierarchy



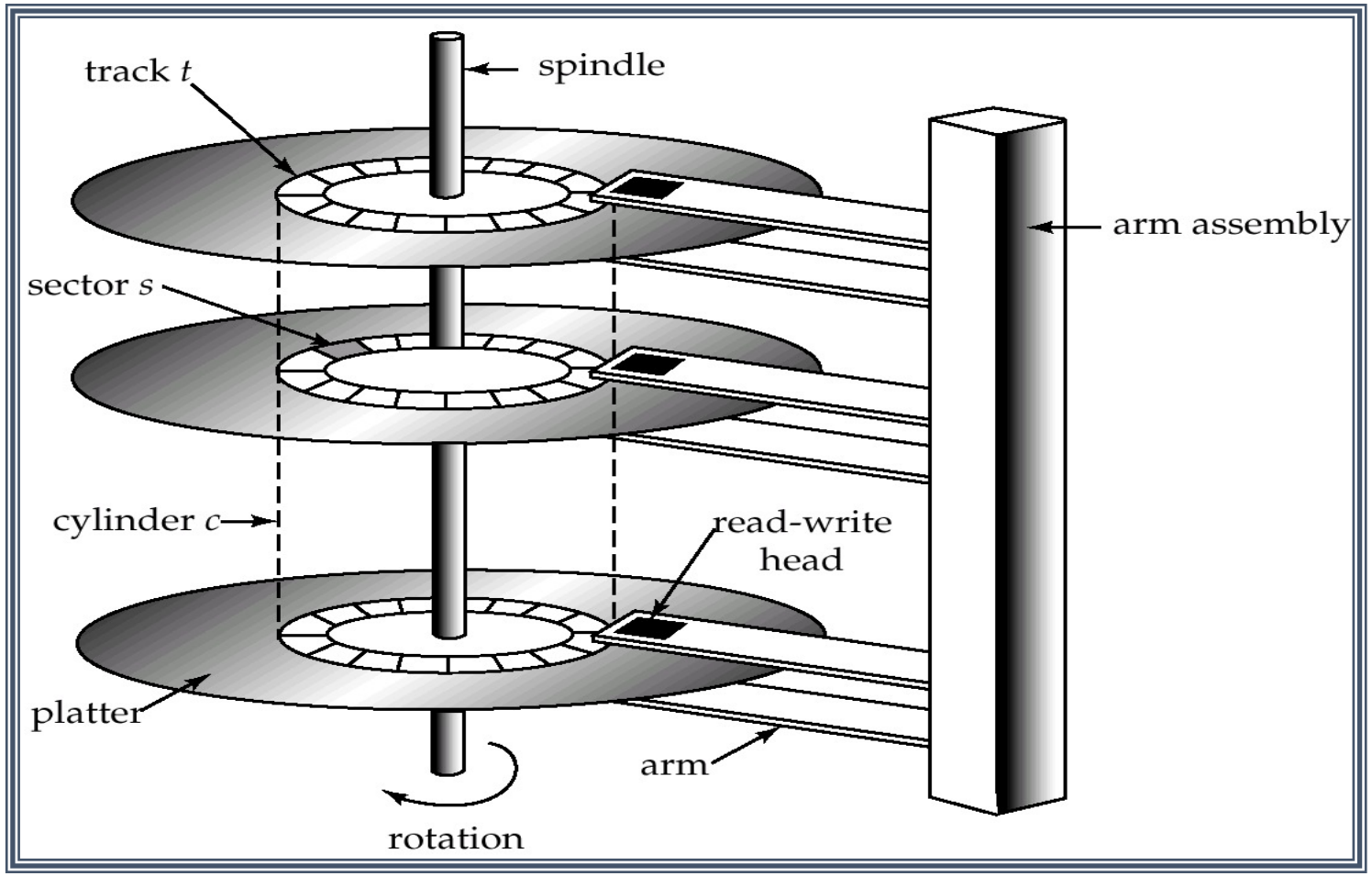
Disk

- 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!

Hard Disk Mechanism



Disk terminology

- Data is stored on disk in units called blocks
- Blocks are arranged in concentric rings called tracks
- Tracks are recorded on one or both surfaces of a platter
- Tracks with the same diameter is called a cylinder; one track per platter surface.
- An array of disk heads one per recorded surface is moved as a unit.

Example

- If a disk has 4 platters and 16K tracks per surface, how many tracks are on the disk?
  - 4,16K,32K,64K,128K?

Terminology

- **Sector** or **Block** – the smallest unit that can be read or written. Often 512 bytes.
- **Track** – all blocks that form a ring on a disk surface that can be read without moving the head.
- **Cylinder** – all tracks on all surfaces, one on top of another, that can be read without moving the head.

Disk Operation

To read (or write) data to the disk:

- The arm containing the read/write heads must be moved to the proper radius from the center.
- The system must wait for the data to rotate under the read head.
- The data is read as it passes under the read head.
- The data is checked and then passed to the I/O controller.

Disk times

- Seek time: Time taken to move the disk head to the track on which the desired block is located
- Rotational delay: Waiting time for the desired block to rotate under the disk head.
  - Time required for half a rotation on average
  - Usually less than seek time
- Transfer time: time to actually read or write the data in the block once the head is positioned.

Disk Time

- Reading or writing a disk block is an I/O operation.
- Time to read or write a block varies, depending on the location of the data:

Access time = seek time + rotational delay + transfer time

Seek time

- Seek Time is fixed by the design of the disk.
- Manufacturers will usually tell you the
  - average seek time
  - maximum seek time *(from center to edge)*
  - time to seek to the next adjacent track.
  - Average: of 4-10ms

Example

- 1 ms to move between cylinders
- 1 ms additional for moving every 4K cylinders
- Move one track: 1.00025 ms
- Move a distance of 65,536 tracks (from outer to inner) = ?

Rotational Delay

- Best case is when the data comes under the head just as it is needed (delay is zero).
- Worst case is you just missed it and have to wait a whole revolution. If you know the rotational speed, you can calculate the time (in ms) per revolution.
- Lets say the disk rotates at 7200 rpm.
- How many rotations per ms?

Rotational Delay Calculation

- 60 seconds per minute.
- 1000 ms per sec
- 1 minute has 60000 ms
- 1 min also 7200 rotations.
- So 1 rotation in 60000/7200 = 8.33ms

How many sectors?

- Files are stored in sectors or blocks on the disk.
- The number of bytes in a sector varies per disk but is often 4096 bytes/sector or 4KB / sector
- Number of Sectors = $\frac{\text{Filesize}}{\text{(bytes/sector)}}$

Transfer time

- The transfer time is determined by how long it takes the data to travel under the head.
- The fraction of sectors on the track that are being read times the rotation time gives the transfer time.
- (Num of sectors) * ms/rotation = transfer_time

Example

- How long does it take to read one sector/block of data on the average?
  - Rotational Speed 10,000 RPM
  - Average Seek Time 4.5 ms
  - Bytes / sector 512
  - Sectors / track 63
- How long does it take to read two sectors?

- Read Example 13.2 from book.

Logical Vs Physical

- Many disks present the OS with a logical layout that is different from the physical layout.
- Most modern disks use Logical Block Addressing (LBA) to hide the physical layout.
- LBA represents the disk as a sequential list of blocks.

Database Storage

- Typically, each table/relation is stored in a separate *file* on the disk
- A file is a logical sequence of blocks.
- Each block consists of a collection of *records*, each corresponding to one row/tuple of the relational model.
- Each record consists of a collection of *fields*, each corresponding to one column/attribute defined in the CREATE TABLE statement.

Database File

- Each record in a DB file has a unqiue record identifier (rid)
- Typically RID = (PageID, SlotNumber) <p,n>
  - Can identify disk address of page containing record by using rid

Example

- Given a Table T with 1M records, size of tuple 20 bytes, and a disk block size of 512bytes, how many disk blocks will T occupy?

Disk Blocks

- Basic unit of data for disk I/O is a *block*
  - Every read/write involves an entire block
- The *blocking factor* (*bfr*) is the number of records that can fit into a block
  - If *B* = bytes per block and *R* = bytes per record, then *bfr* = floor(*B*/*R*), as long as each record is contained entirely in one block
  - For records that are divided among blocks, *bfr* is just the average number of records in each block

Terminology: Pages Vs Blocks

- Each device may support a different block size
- Pages are virtual blocks.
- With pages the OS can deal with a fixed size page, rather than try to figure out how to deal with blocks of different sizes.
- Pages act as sort of a middleman between operating systems and hardware drivers

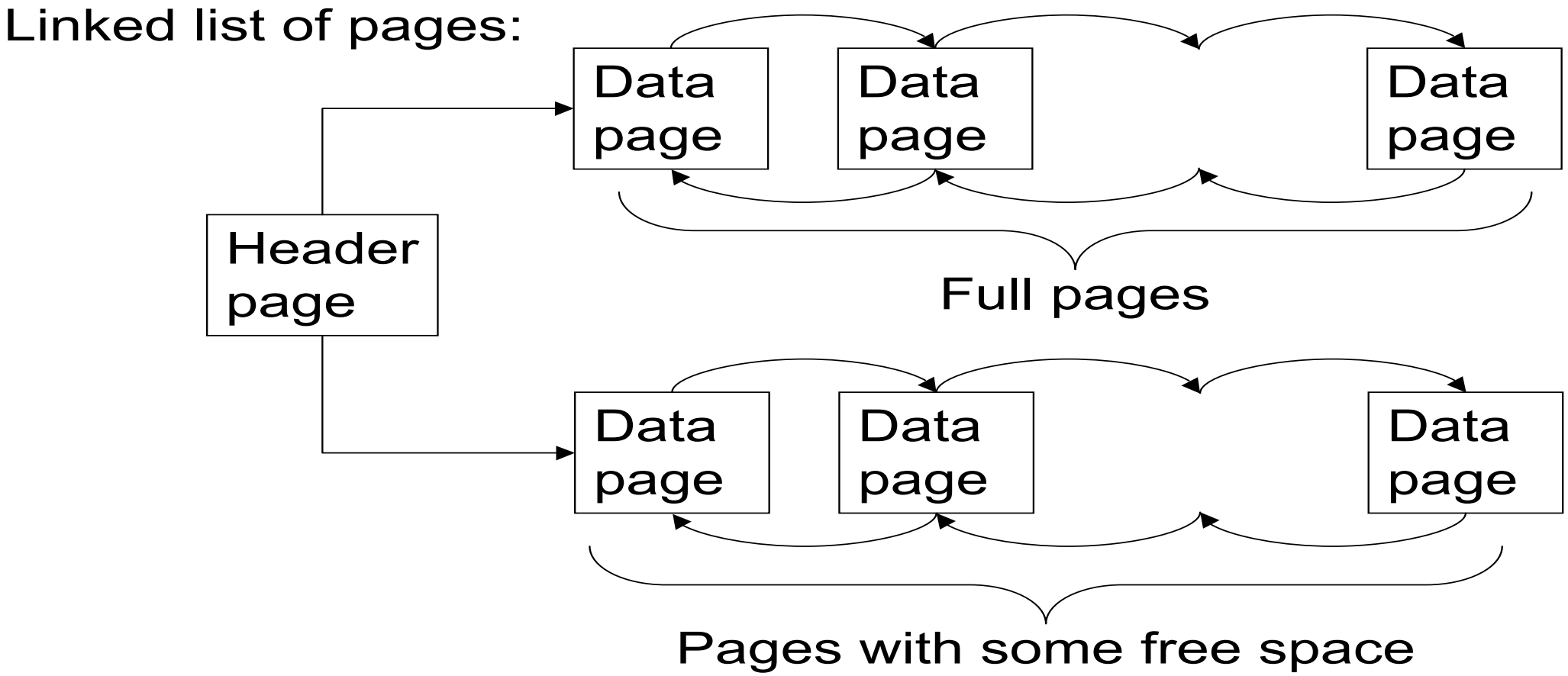
Unordered Files (Heap File)

- No particular order maintained on the records
- Pages no control anyways.
- Unordered collection of records
- Each record in a heap file has a unique record identifier (rid)
- Typically, RID = (PageID, SlotNumber) <p,n>
  - Can identify disk address of page containing record by using rid

Unordered Files (Heap File)

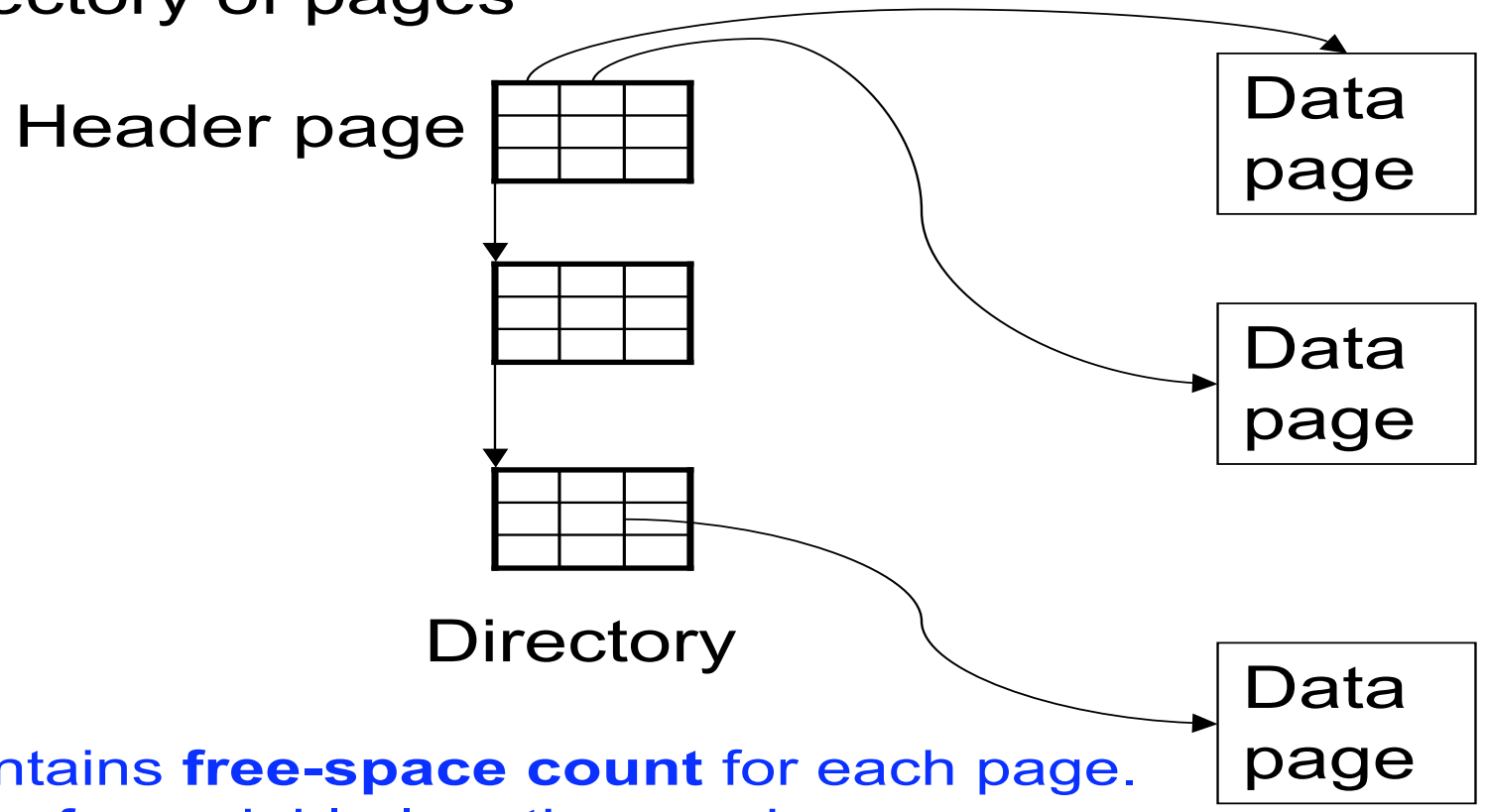
- No particular order maintained on the records
  - Insertions done at the end of the file (O(1))
  - Deletions can be done efficiently provided you know the row to delete. Then (O(1))
    - move last element in file to replace deleted element
  - Searching for a record needs linear search (O(n))
    - n/2 records read on average, n in worst case
  - Updating a record may be costly also (O(n))
    - O(1) if you do not have to search for the record

Heap File Implementation 1



Heap File Implementation 2

Better: directory of pages



Directory contains **free-space count** for each page.
Faster inserts for variable-length records

Ordered Files

- Records are stored in order based on the value of an *ordering field*

Ordered Files

- Records are stored in order based on the value of an *ordering field*
  - Searches are faster, using binary search
    - O(log n) steps in worst case, for ordering field only
    - No benefit for accesses on other fields (O(n))
  - Inserts, deletes, and updates are slower in the worst case, since order must be maintained (O(n))
- Oracle does not support ordered files, but some other DBMSs do

Record field type

- Record field type defines their length
  - Fixed-length e.g., INTEGER, BIGINT, CHAR(n), DATE,…
  - Variable –length: VARCHAR(n), CLOB(n),…

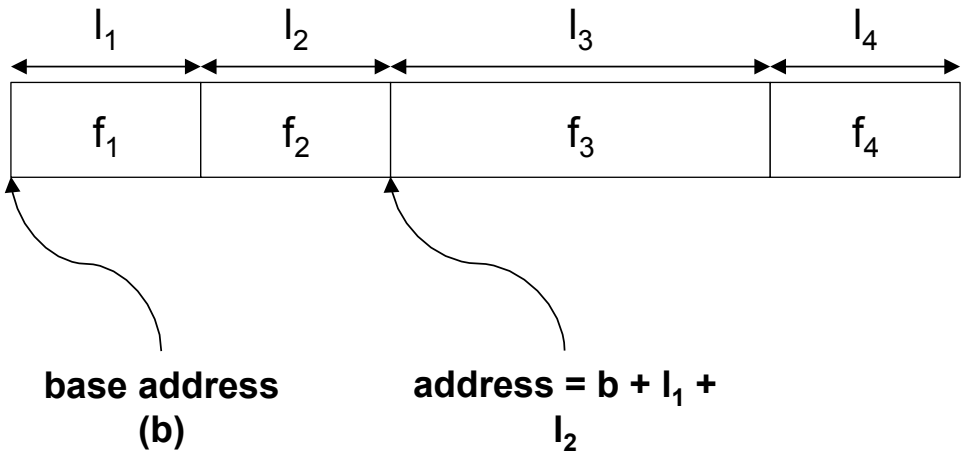
Page Formats

- Issues to consider
  - 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
  - Composed of **fixed** length fields
  - Composed off **variable** length fields

Record field type

- Record field type defines their length
  - Fixed-length e.g., INTEGER, BIGINT, CHAR(n), DATE,…
  - Variable –length: VARCHAR(n), CLOB(n),…

Fixed-length fields

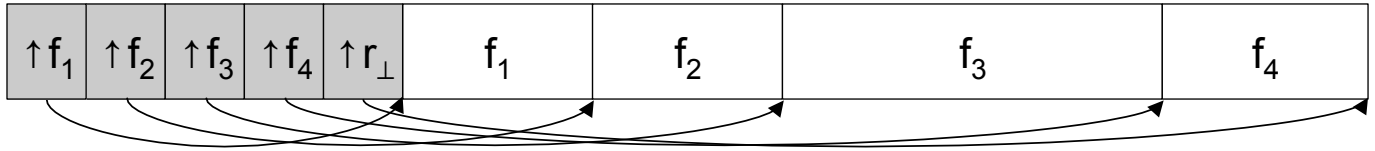


- Fixed-length record: each field has a **fixed length** and the number of fields is also **fixed**
  - fields can be stored **consecutively**
  - given the address of the record (b), the address of a particular field can be calculated using information about **lengths of preceding fields** (l_i)
  - this information is available from the DBMS **system catalog**

Variable-length Fields

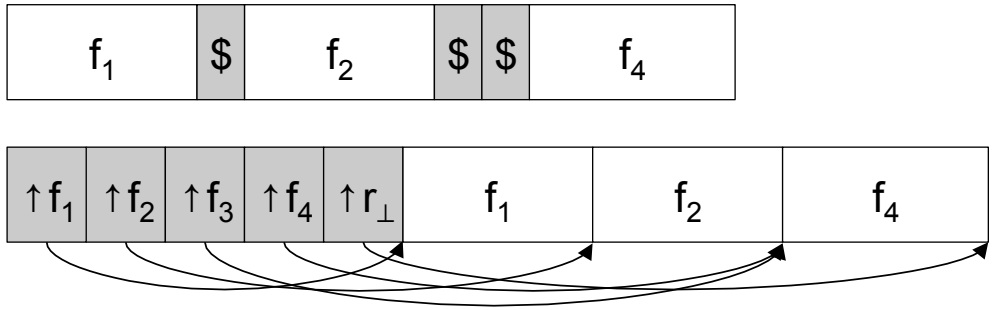
- Multiple variants exist to store records that contain variable- length fields
  1. use a special **delimiter symbol** ($) to separate record fields: accessing field f_n requires a scan over the bytes of fields $f_1 \ldots f_{n-1}$

  2. for a record of *n* fields, use an **array** of *n* + 1 offsets pointing into the record (the last array entry marks the end of field f_n)



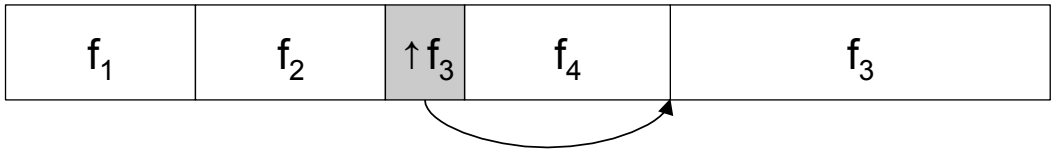
Delimiter Vs Array

- Array approach is typically superior
  - array overhead translates to **direct access** to any field
  - clean and compact way to deal with **null values** (`NULL` in SQL) by simply comparing pointers to **beginning** and **end** of field



Record Formats

- Another popular record format to support variable-length fields is a **combination** of the delimiter and array approach
  - variable-length fields are stored **at the end of the record**
  - fixed-length fields and pointers to variable-length fields are stored sequentially, starting **at the beginning of the record**

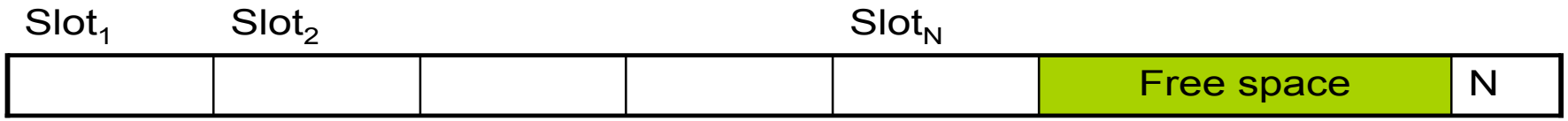


Page Formats

- Issues to consider
  - 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
  - Composed of **fixed** length fields
  - Composed off **variable** length fields

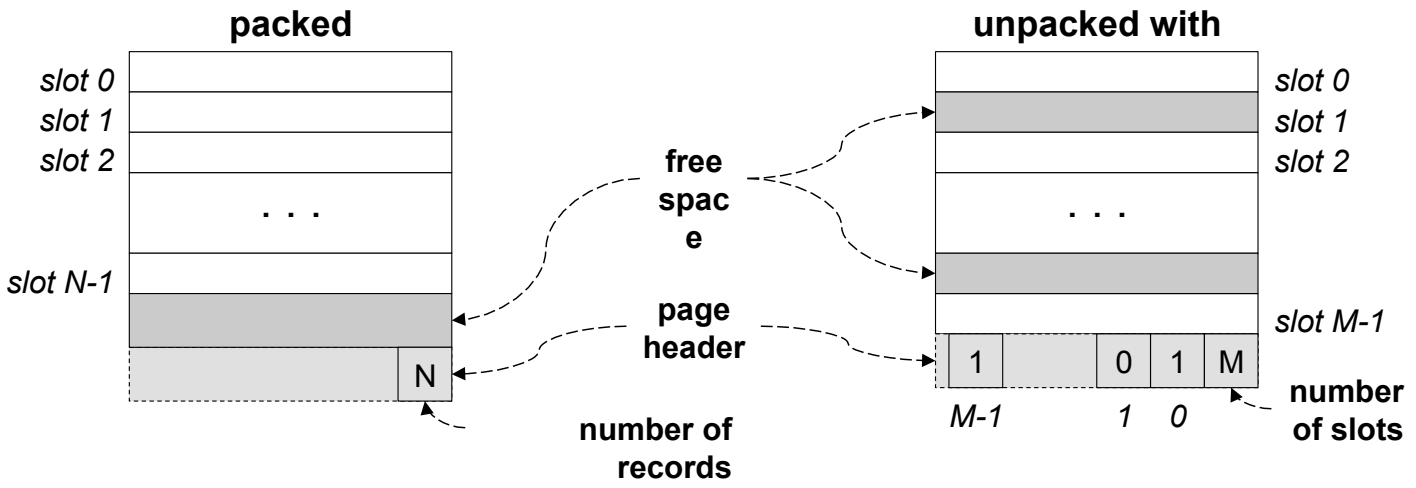
Page Formats Approach 1

- Fixed-length records: packed representation



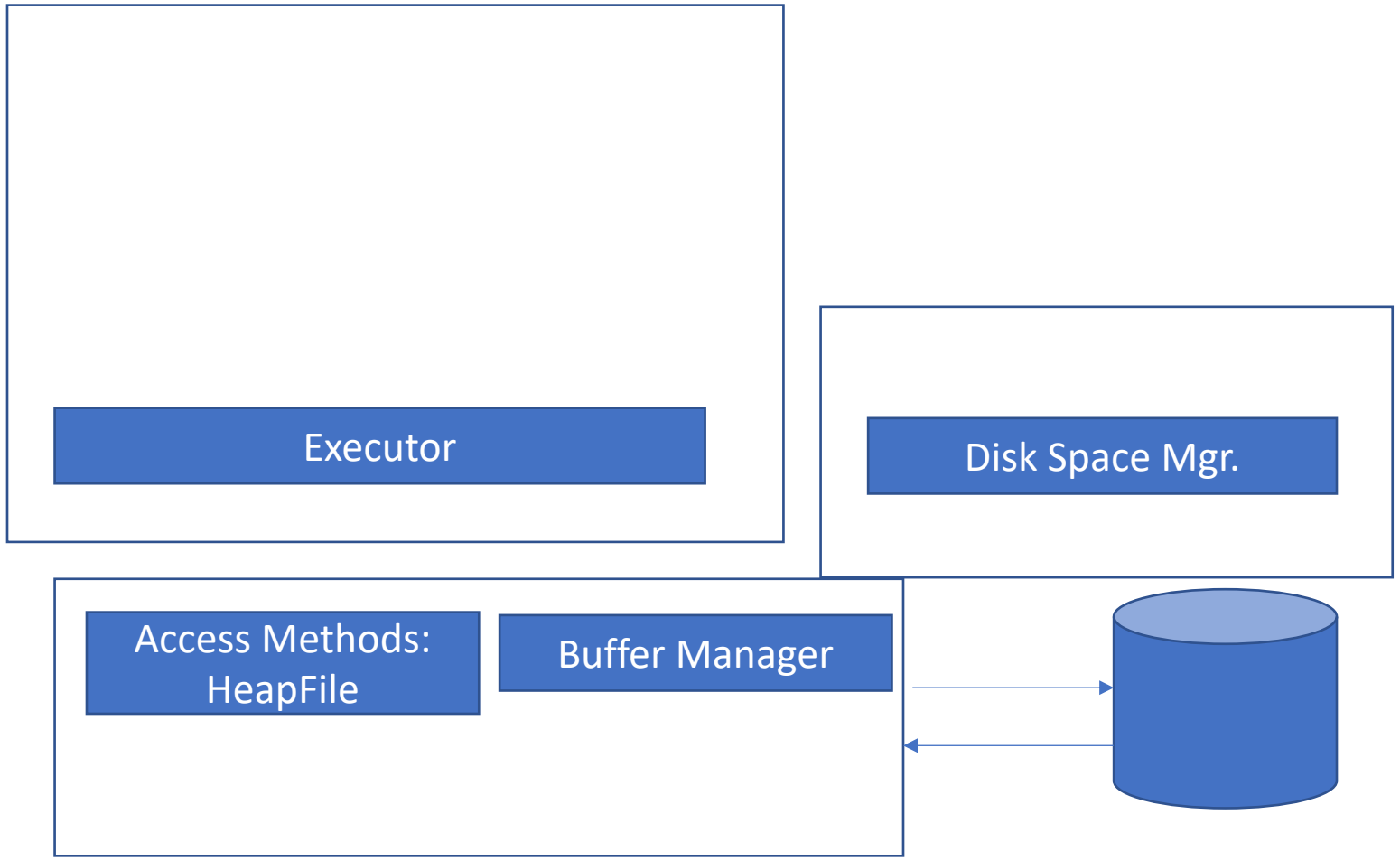
- All records on the page (in the file) are the **same size** *s*
  - **`getRecord(`***f*, <*p*, *n*>**`)`** : given the *rid* <*p*, *n*> we know that the record is to be found at (byte) offset *n* × *s* on page *p*
  - **`deleteRecord(`***f*, <*p*, *n*>**`)`** : copy the bytes of the last occupied slot on page *p* to offset *n* × *s*, mark slot as free (page is **packed**, i.e., all occupied slots appear together at the start if the page)
  - **`insertRecord(`***f*, *r***`)`** : find a page *p* with free space >= *s* (see previous discussion) and copy *r* to the first free slot on *p*, then mark the slot as occupied

Solution1: Free Slot Bitmap



- Avoid record copying and therefore rid modifications
  - **`deleteRecord(`***f*, <*p*, *n*>**`)`** simply needs to set bit *n* in bitmap to 0
  - **no other** *rids* affected

Storage Manager



- Process data with a simple query.
- Access methods: Organize data to support fast access to desired subsets of records.
- Buffer manager: Caches data in memory. Reads/writes data to/from disk as needed
- Disk-space manager: Allocates space on disk for files/access methods

Spatial Control (Using raw disk interface)

- Overview
  – DBMS issues low-level storage requests directly to disk device
- Advantages
  - DBMS can ensure that important queries access data sequentially
  - Can provide highest performance
- Disadvantages
  - Requires devoting entire disks to the DBMS
  - Reduces portability as low-level disk interfaces are OS specific

Spatial Control (Using OS Files)

- **Overview**
  - DBMS creates one or more very large OS files
- **Advantages**
  - Allocating large file on empty disk can yield good physical locality
- **Disadvantages**
  - OS can limit file size to a single disk
  - OS can limit the number of open file descriptors
  - But these drawbacks have mostly been overcome by modern OSs

Historical Perspective (1981)

- Recognizes mismatch problem between OS files and DBMS needs
  - If DBMS uses OS files and OS files grow with time, blocks get scattered
  - OS uses tree structure for files but DBMS needs its own tree structure
- Other proposals at the time
  - Extent-based file systems
  - Record management inside OS

Commercial Systems

- Most commercial systems offer both alternatives
  - Raw device interface for peak performance
  - OS files more commonly used
- In both cases, we end-up with a DBMS file abstraction implemented on top of OS files or raw device interface

Temporal Control (Buffer Manager)

- Correctness problems
  - DBMS needs to control when data is written to disk in order to provide **transactional semantics** (we will study transactions later)
  - OS buffering can **delay writes**, causing problems when crashes occur
- Performance problems
  - OS optimizes buffer management for general workloads
  - DBMS understands its workload and can do better
  - Areas of possible optimizations
    - Page replacement policies
    - Read-ahead algorithms (physical vs logical)
    - Deciding when to flush tail of write-ahead log to disk

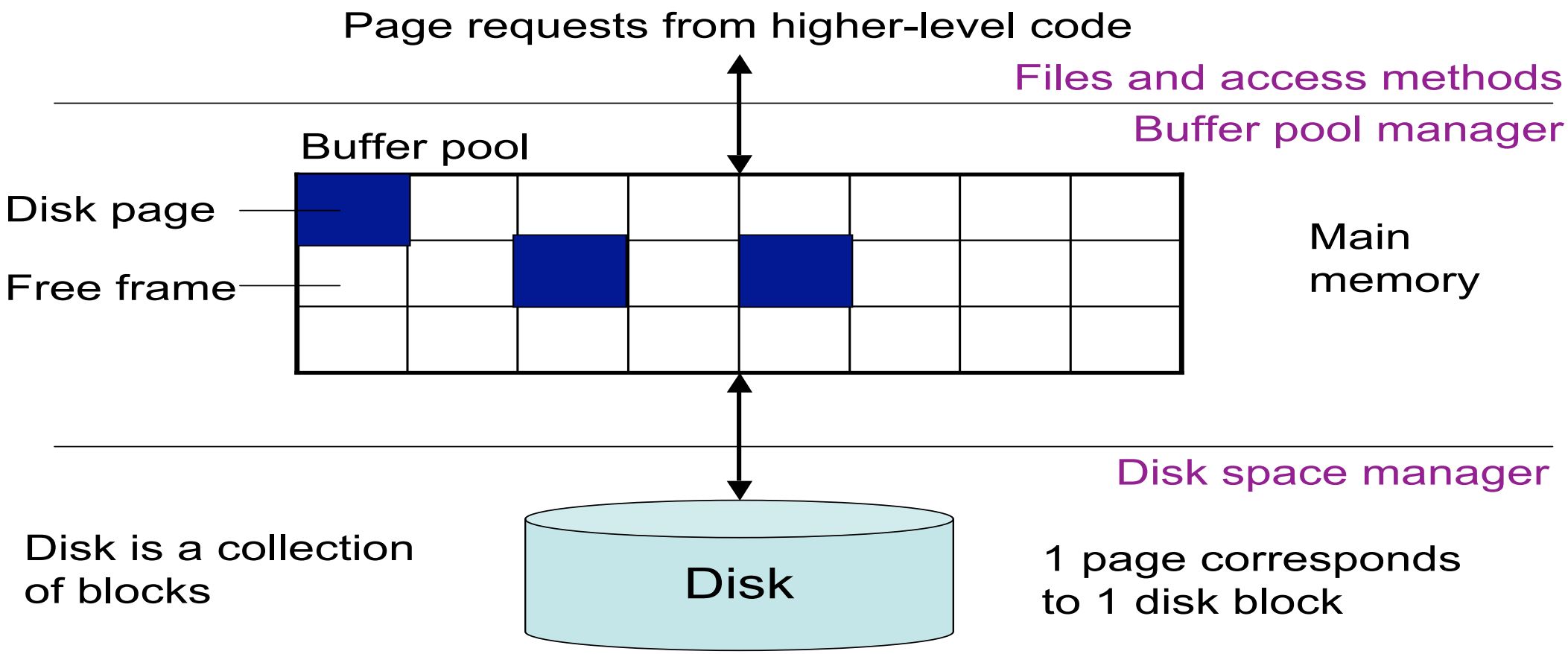
Historical Perspective (1981)

- Problems with OS buffer pool management long recognized
  - Accessing OS buffer pool involves an expensive system call
  - Faster to access a DBMS buffer pool in user space
  - LRU replacement does not match DBMS workload
  - DBMS can do better
  - OS can do only sequential prefetching, DBMS knows which page it needs next and that page may not be sequential
  - DBMS needs ability to control when data is written to disk

Commercial Systems

- DBMSs implement their own buffer pool managers
- Modern file systems provide good support for DBMSs
  - Using large files provides good spatial control
  - Using interfaces like the mmap suite
    - Provides good temporal control
    - Helps avoid double-buffering at DBMS and OS levels

Buffer Manager



SimpleDB

- Demo Git/Eclipse Setup
- Go through an overview of SimpleDB

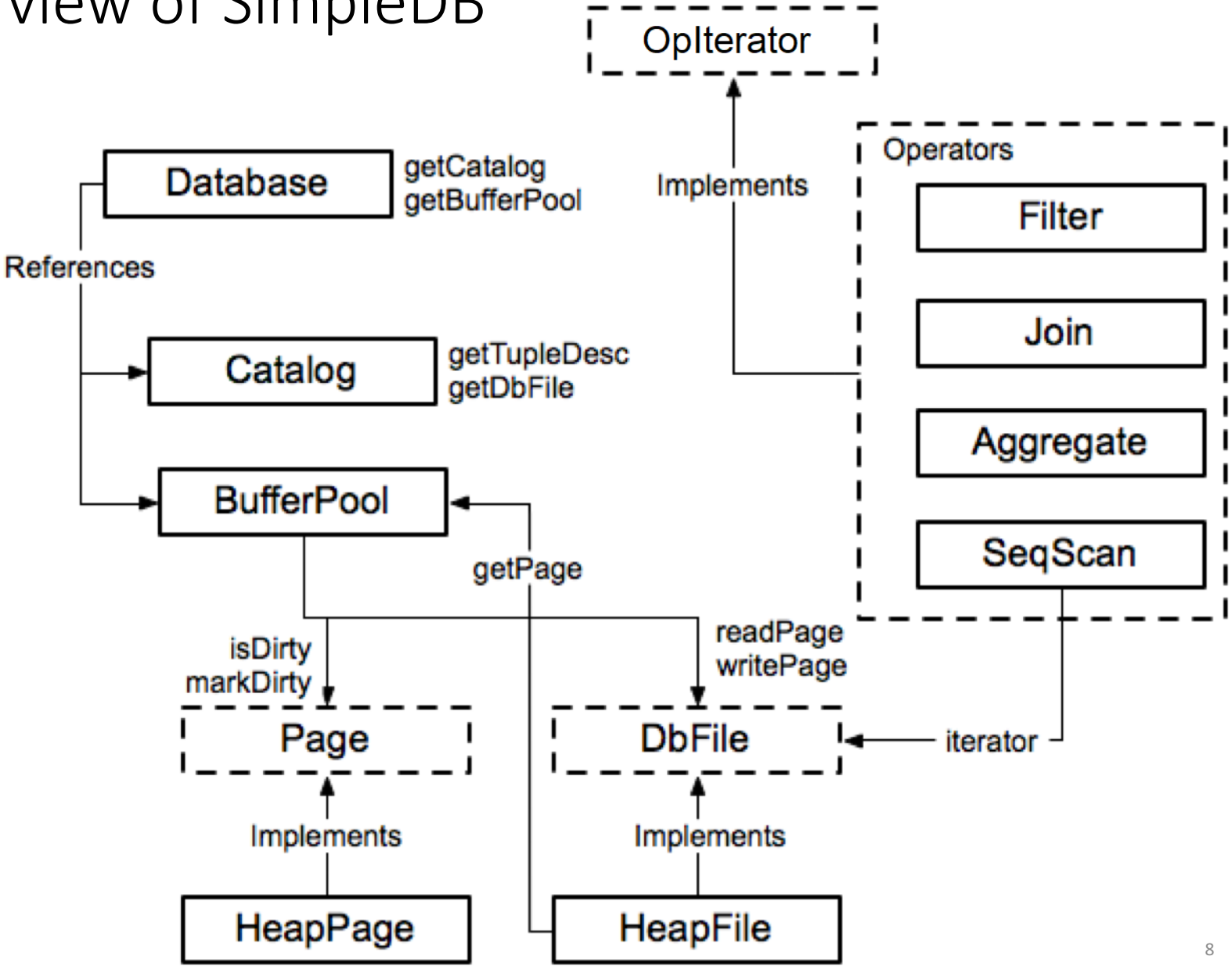
What you should NOT do

- Modify given classes
- Removing, renaming, relocating to other packages
- Modify given methods
- Changing parameters or return types
- Use third-party libraries
- Except the ones under lib/directory
- You can do everything using regular Java libraries

What you can do

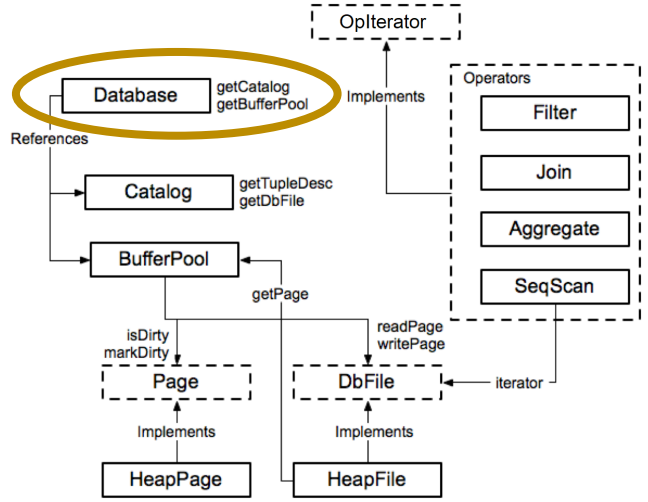
- Add code to existing classes/methods
  - Re-implement provided methods
- Run system test cases
  - Under test/systemtest
  - I'll grade using additional tests
- Add new classes/interfaces/methods/packages
  - Haven't found the need to do for Lab 1 and 2.
- Just don't destroy correctness or specification!
- Find bugs!
- Write up
- Explain why do you implement in a particular way
  - I'll read your code
  - Reading horrible code is horrible, so spend some time polishing
  - Passing all the test cases may not necessary mean you'll get a high score

Overview of SimpleDB



Database

- A single database
  - One schema
  - List of tables
- References to major components
  - Global instance of Catalog
  - Global instance of BufferPool



Catalog

- Stores metadata about tables in the database
  - void addTable(DbFile d, TupleDesc d)
  - DbFile getTable(int tableid)
  - TupleDesc getTupleDesc(int tableid) •...
- NOT persisted to disk
  - Catalog info is reloaded every time SimpleDB starts up