# Before we talk about Big Data…

# Lets talk about not-so-big data

# Brief Intro to Database Systems

Tova Milo, milo@cs.tau.ac.il

# Textbook(s)

**Main textbook (In the library)**

- *Database Systems: The Complete Book*, Hector Garcia-Molina, Jeffrey Ullman, Jennifer Widom

**Almost identical**

- *A First Course in Database Systems*, Jeff Ullman and Jennifer Widom
- *Database Implementation*, Hector Garcia-Molina, Jeff Ullman and Jennifer Widom

# What is a (Relational) Database Management System ?

Database Management System = **DBMS**

Relational DBMS = **RDBMS**

- A collection of files that store the data
- A big C program written by someone else that accesses and updates those files for you

# Example of a Traditional Database Application

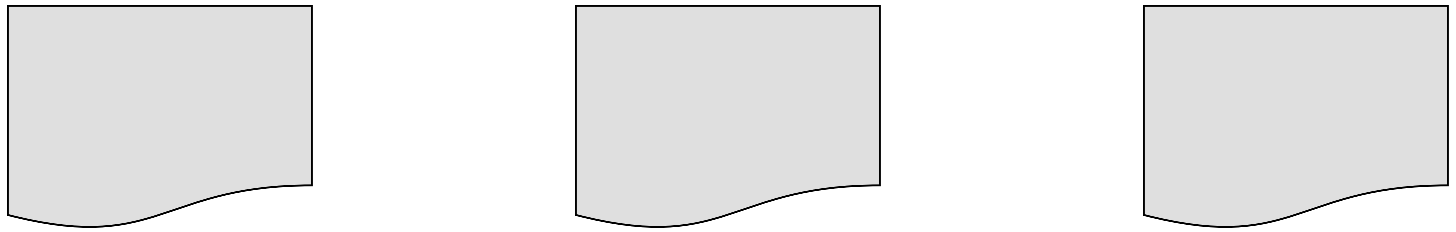
Suppose we are building a system to store the information about:

- students
- courses
- professors
- who takes what, who teaches what

# Can we do it without a DBMS ?

Sure we can! Start by storing the data in files:

students.txt courses.txt professors.txt



Now write C or Java programs to implement specific tasks

# Doing it without a DBMS...

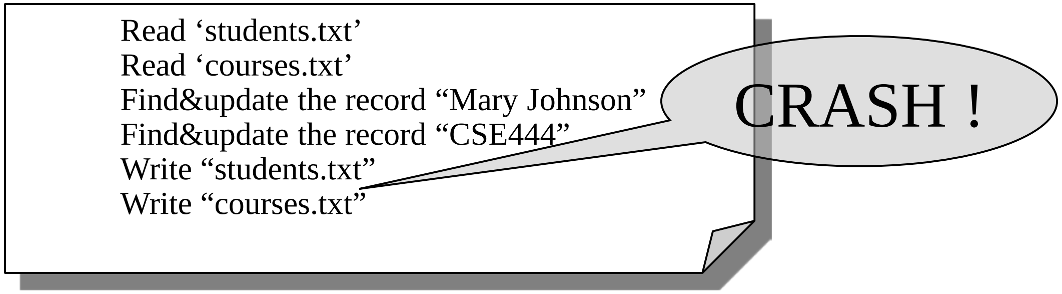
- Enroll "Mary Johnson" in "CSE444":

Write a C program to do the following:

Read 'students.txt'
Read 'courses.txt'
Find&update the record "Mary Johnson"
Find&update the record "CSE444"
Write "students.txt"
Write "courses.txt"

# Problems without an DBMS...

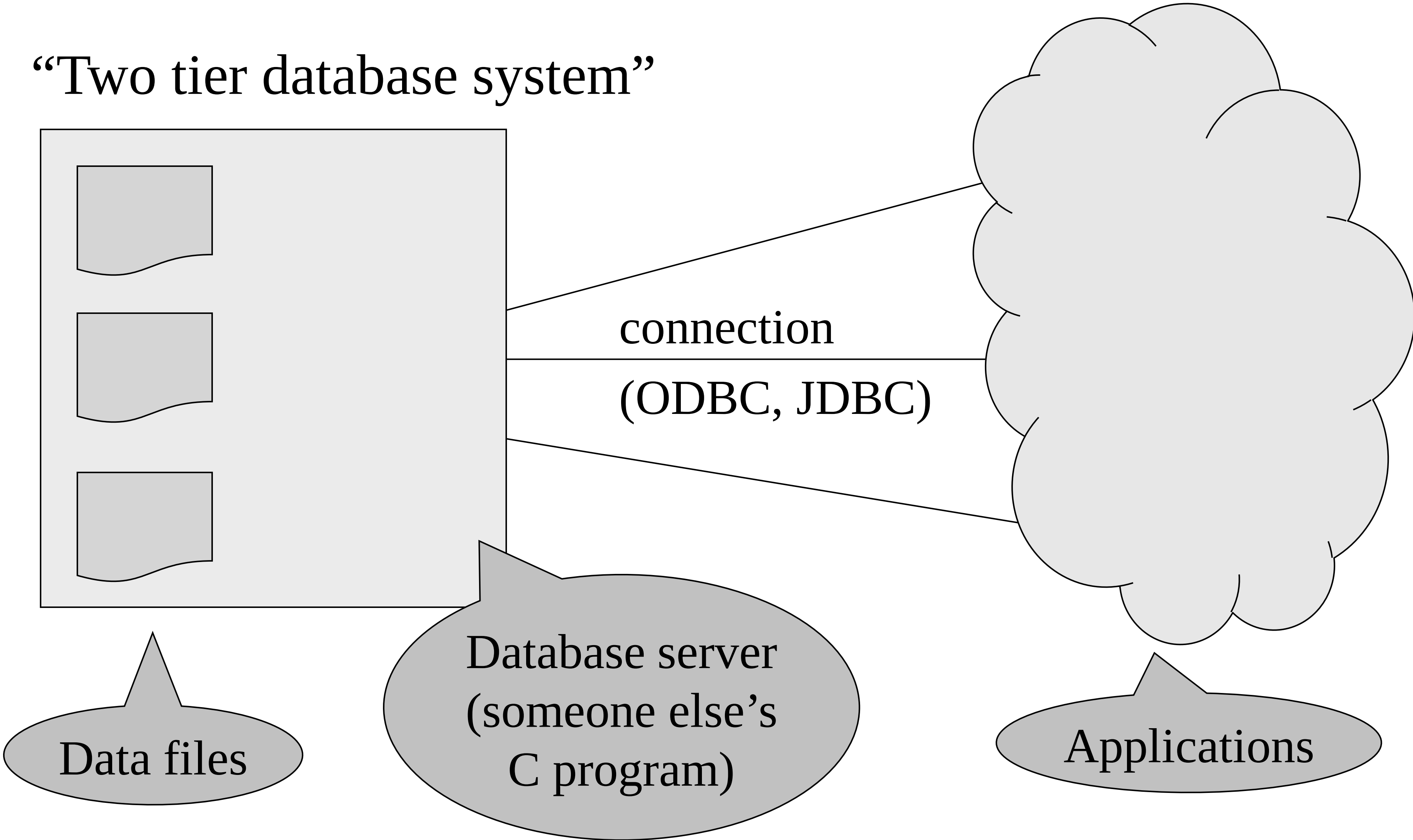
- System crashes:




  – What is the problem ?
- Large data sets (say 50GB)
  – What is the problem ?
- Simultaneous access by many users
  – Need locks: we know them from OS, but now data on disk; and is there any fun to re-implement them ?

# Enters a DMBS

"Two tier database system"



# How the Programmer Sees the DBMS

- Tables:

Students:

| SSN | Name | Category |
|---|---|---|
| 123-45-6789 | Charles | undergrad |
| 234-56-7890 | Dan | grad |
| | … | … |

Takes:

| SSN | CID |
|---|---|
| 123-45-6789 | CSE444 |
| 123-45-6789 | CSE444 |
| 234-56-7890 | CSE142 |
| | … |

Courses:

| CID | Name | Quarter |
|---|---|---|
| CSE444 | Databases | fall |
| CSE541 | Operating systems | winter |

- Still implemented as files, but behind the scenes can be quite complex

***"data independence"*** = separate *logical* view from *physical implementation*

# Queries

- Find all courses that “Mary” takes

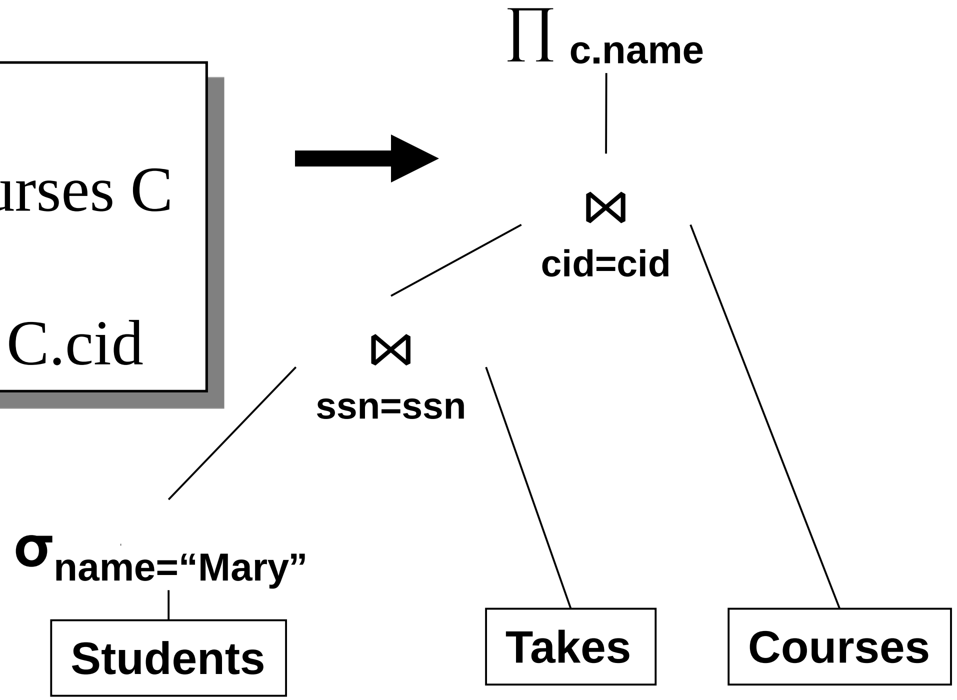
```
SELECT  C.name
FROM    Students S, Takes T, Courses C
WHERE   S.name="Mary" and
        S.ssn = T.ssn and T.cid = C.cid
```

- What happens behind the scene ?
  - Query processor figures out how to answer the query efficiently.

# Queries, behind the scene

***Declarative SQL query*** → ***Imperative query execution plan:***

```
SELECT  C.name
FROM Students S, Takes T, Courses C
WHERE S.name="Mary" and
        S.ssn = T.ssn and T.cid = C.cid
```




The **optimizer** chooses the best execution plan for a query

Table name

Attribute names

# Tables in SQL

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

Tuples or rows

# SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  attributes
FROM    relations (possibly multiple)
WHERE   conditions (selections)
```

# Simple SQL Query

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

| PName | Price | Manufacturer |
|---|---|---|
| SingleTouch | $149.99 | Canon |
| MultiTouch | $203.99 | Hitachi |

"selection" and "projection"

# Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='gizmo' AND price > 50
ORDER BY price, pname
```

Ordering is ascending, unless you specify the DESC keyword.

Ties are broken by the second attribute on the ORDER BY list, etc.

# Joins in SQL (1)

- Connect two or more tables:

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

Company

| Cname | StockPrice | Country |
|---|---|---|
| GizmoWorks | 25 | USA |
| Canon | 65 | Japan |
| Hitachi | 15 | Japan |

What is the connection between them ?

# Joins in SQL (2)

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

Join between Product and Company

```
SELECT   pname, price
FROM     Product, Company
WHERE    manufacturer=cname AND country='Japan'
         AND price <= 200
```

# Joins in SQL (3)

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

Company

| Cname | StockPrice | Country |
|---|---|---|
| GizmoWorks | 25 | USA |
| Canon | 65 | Japan |
| Hitachi | 15 | Japan |

```
SELECT   pname, price
FROM     Product, Company
WHERE    manufacturer=cname AND country='Japan'
         AND price <= 200
```

| PName | Price |
|---|---|
| SingleTouch | $149.99 |

# Grouping and Aggregation

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

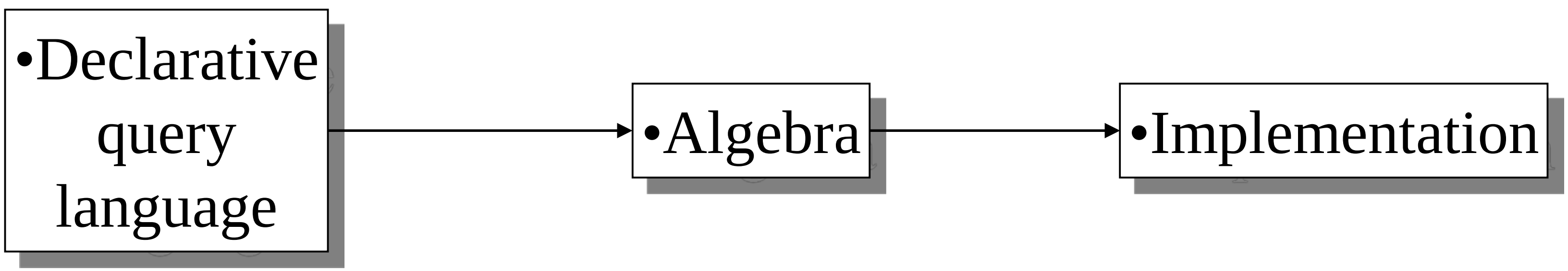
Find all products under $200 manufactured in Japan;
return their names and (for each name) the average price.

```
SELECT   pname, AVG(price)
FROM     Product, Company
WHERE    manufacturer=cname AND country='Japan'
         AND price <= 200

GROUP BY  pname
```

# Relational Algebra

- Its place in the big picture:




•SQL

•Relational algebra

# Relational Algebra

- Operators
  - Selection: $\sigma_{A=123}(R)$
  - Projection: $\Pi_{A,B}(R)$
  - Join: $R \bowtie_{\theta} S$
  - Group: $\gamma_{A,sum(B)}(R)$
  - …

# The query from before

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and average price.

```
SELECT   pname, AVG(price)
FROM     Product, Company
WHERE    manufacturer=cname AND country='Japan'
         AND price <= 200

GROUP BY  pname
```

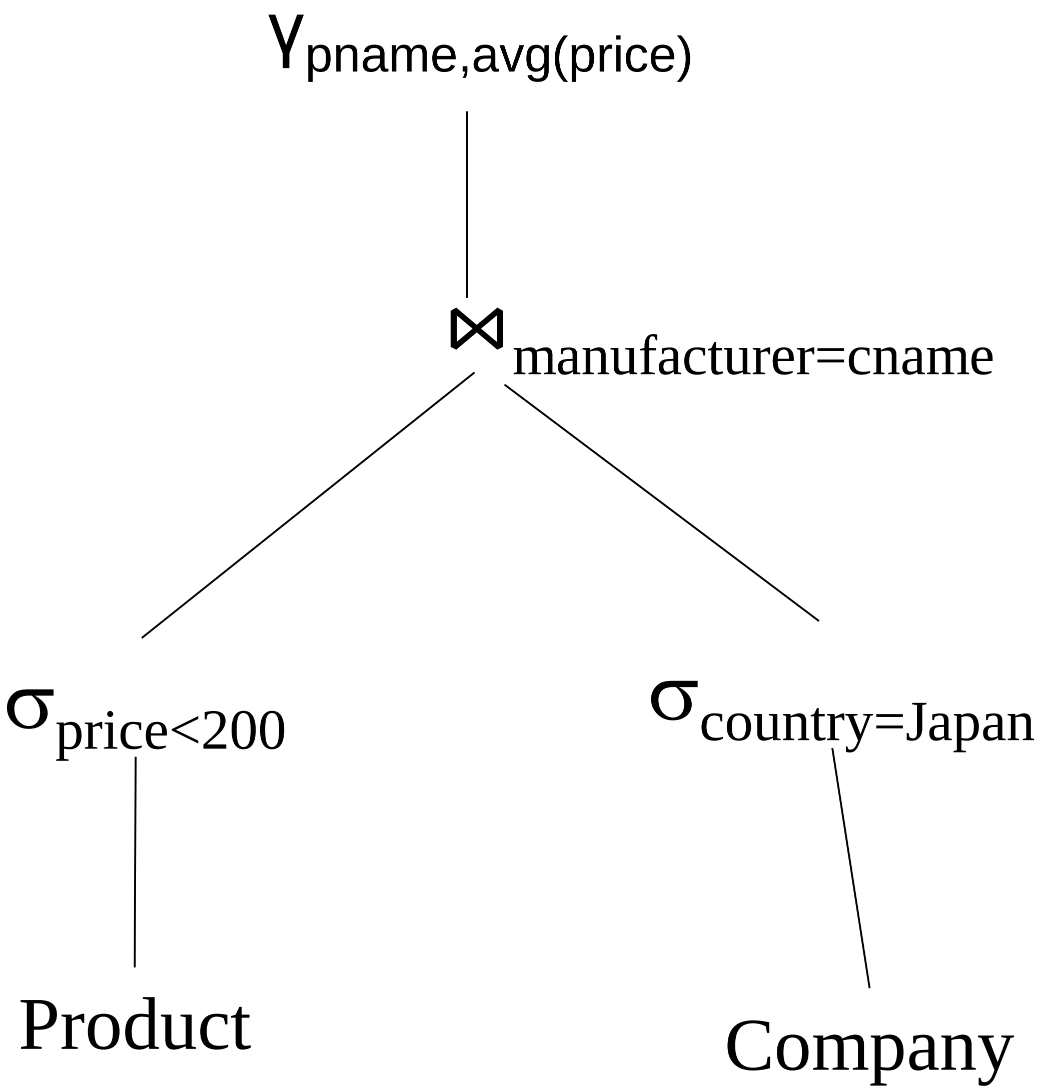
# The same query in algebra

$$\gamma_{pname,avg(price)}[\sigma_{price<=200}(Product)) \bowtie_{manufacturer=cname} (\sigma_{country=Japan}(Company))]$$

```
SELECT   pname, AVG(price)
FROM     Product, Company
WHERE    manufacturer=cname AND country='Japan'
         AND price <= 200

GROUP BY pname
```

# Tree-shaped version



# Single Node Query Processing

Given relations R(A,B) and S(B, C), no indexes:

- Selection: $\sigma_{A=123}(R)$
  - Scan file R, select records with A=123
- Group-by: $\gamma_{A,sum(B)}(R)$
  - Scan file R, insert into a hash table using attr. A as key
  - When a new key is equal to an existing one, add B to the value
- Join: $R \bowtie S$
  - Scan file S, insert into a hash table using attr. B as key
  - Scan file R, probe the hash table using attr. B