COMP 440: Database Design

Lecture 1: Introduction

Mahdi Ebrahimi

data is the new oil
BIG DATA
CLOUD
SECURITY
DEVELOPERS
AZURE
APPS
PLATFORM
WEB
DEVICES
PEOPLE
memory
MAGIC

NATIONAL CANCER INSTITUTE
GENOMIC DATA COMMONS

Access the Data

#NCIGDC

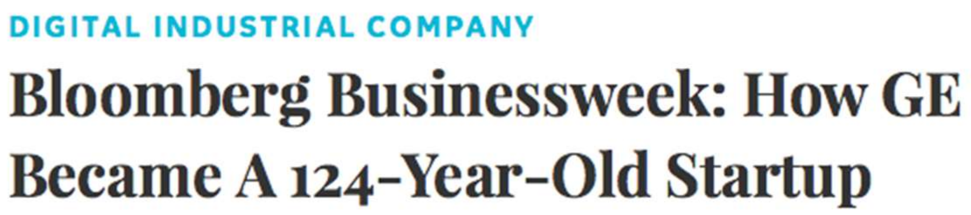
elixir

ELIXIR Innovation and SME Forum:
Data-driven innovation in food, nutrition and the microbiome

Big science is data driven.

Increasingly many companies see themselves as data driven.

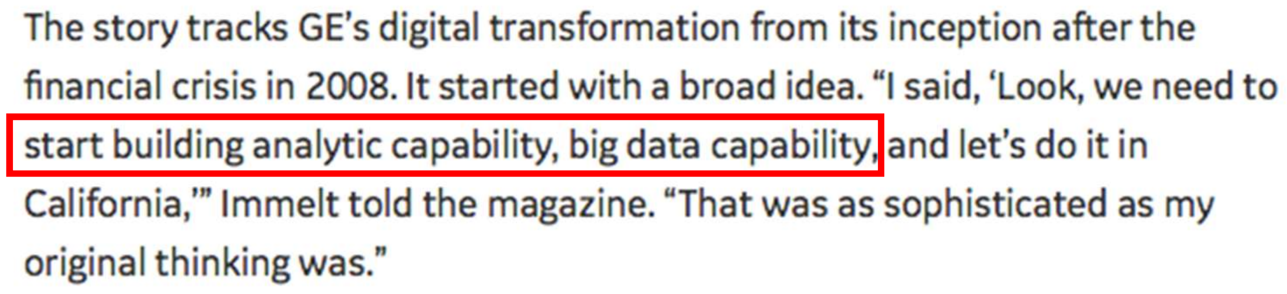
Even more “traditional” companies…



DIGITAL INDUSTRIAL COMPANY

Bloomberg Businessweek: How GE Became A 124-Year-Old Startup

The digital power plant is one of Industrial Internet applications. Image credit: GE Power



The story tracks GE’s digital transformation from its inception after the financial crisis in 2008. It started with a broad idea. “I said, ‘Look, we need to start building analytic capability, big data capability, and let’s do it in California,’” Immelt told the magazine. “That was as sophisticated as my original thinking was.”

The world is increasingly driven by data…

This class teaches the basics of how to use & manage data.

Today’s Lecture

1. Introduction

2. Overview of the relational data model

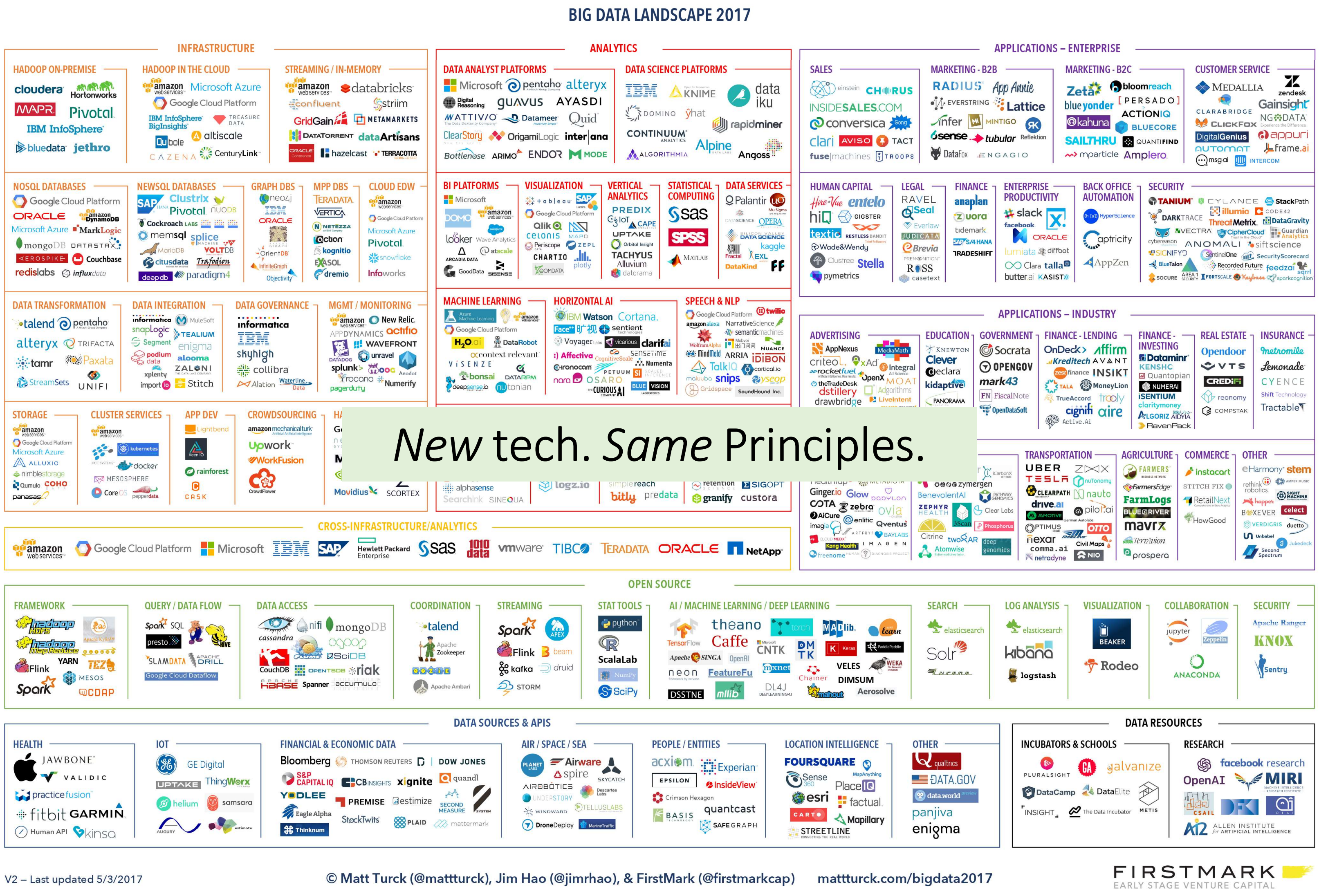
3. Overview of DBMS topics: Key concepts & challenges

1. Introduction

What you will learn about in this section

1. Motivation for studying DBs

2. Overview of lecture coverage


BIG DATA LANDSCAPE 2017
INFRASTRUCTURE
ANALYTICS
APPLICATIONS – ENTERPRISE
APPLICATIONS – INDUSTRY
CROSS-INFRASTRUCTURE/ANALYTICS
OPEN SOURCE
DATA SOURCES & APIS
DATA RESOURCES
V2 – Last updated 5/3/2017
© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark (@firstmarkcap)
mattturck.com/bigdata2017
FIRSTMARK EARLY STAGE VENTURE CAPITAL

New tech. *Same* Principles.

Why should you study databases?

- **Mercenary-make more $$$:**
  - Startups need DB talent right away = low employee #
  - Massive industry…

- **Intellectual**:
  - Science: data poor to data rich
    - No idea how to handle the data!
  - Fundamental ideas to/from all of CS:
    - Systems, theory, AI, logic, stats, analysis….

Many great computer systems ideas started in DB.

What this course is (and is not)

- Discuss **fundamentals of data management**
  - How to design databases, query databases, build applications with them.
  - How to debug them when they go wrong!
  - <u>Not</u> how to be a DBA or how to tune Oracle 12g.
- We'll cover **how database management systems work**
- And some **basic principles of how to build** them

Lectures: 1st part - from a user's perspective

1. **Foundations:** Relational data models & SQL
   - How to manipulate data with SQL, a declarative language
     - *reduced expressive power but the system can do more for you*

2. **Database Design**: Design theory and constraints
   - Designing relational schema to keep your data from getting corrupted

Lectures: 2nd part – database internals

3. Introduction to database systems

- Data Storage and IO models
- File Organization

4. Indexing and Hashing

- Intro to indexing
- B+ Tree, Hash Indexes

5. Query processing

- Access methods and operators
- Joins
- Relational algebra and Query optimization

Lectures: 3rd part – NoSQL Databases

6. NoSQL Databases

2. Overview of the relational data model

What you will learn about in this section

1. Definition of DBMS
2. Data models & the relational data model
3. Schemas & data independence

What is a DBMS?

- A large, integrated collection of data
- Models a real-world <u>*enterprise*</u>
  - *Entities* (e.g., Students, Courses)
  - *Relationships* (e.g., Alice is enrolled in COMP440)

A <u>Database Management System (DBMS)</u> is a piece of software designed to store and manage databases

A Motivating, Running Example

- Consider building a course management system (**CMS**):
  - Students
  - Courses
  - Professors

  Entities

  - Who takes what
  - Who teaches what

  Relationships

Data models

- A **data model** is a collection of concepts for describing data
  - The relational model of data is the most widely used model today
    - Main Concept: the *relation*- essentially, a table
- A **schema** is a description of a particular collection of data**, using the given data model**
  - E.g. every *relation* in a relational data model has a *schema* describing types, etc.

Modeling the Course Management System

- *Logical Schema*
  - Students(sid: *string*, name: *string*, gpa: *float*)
  - Courses(cid: *string*, cname: *string*, credits: *int*)
  - Enrolled(sid: *string,* cid*: string,* grade*: string*)

Relations

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

sid	cid	Grade
123	564	A

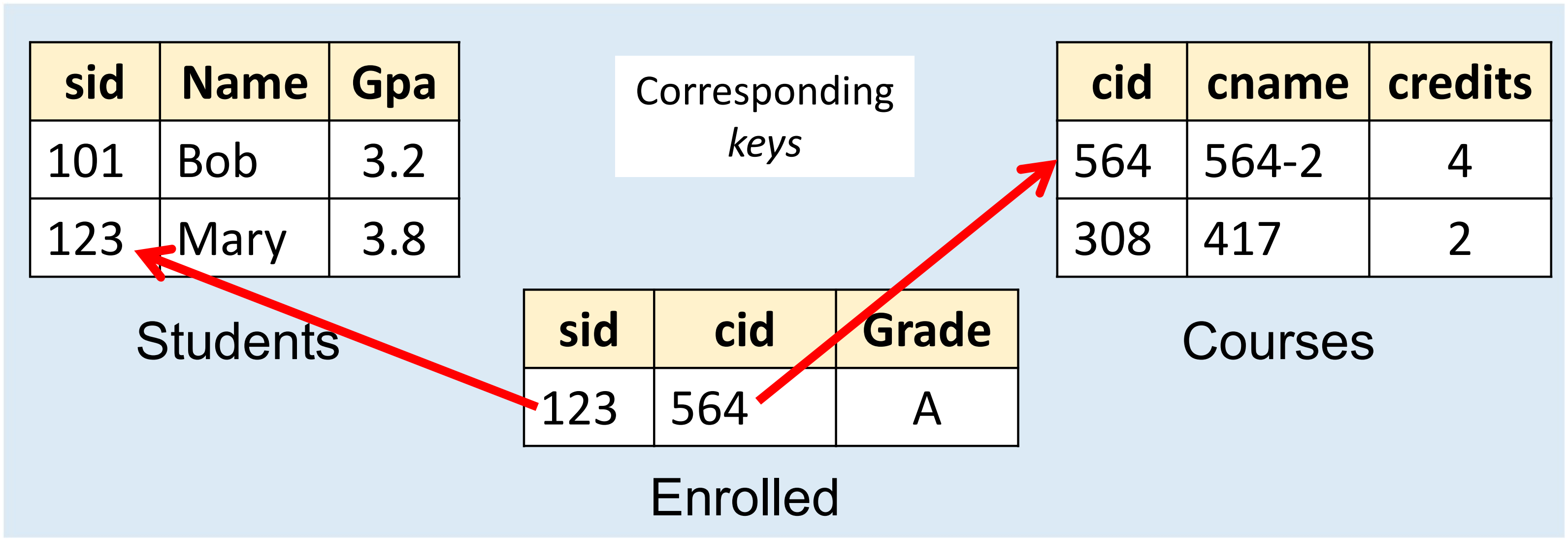
Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Modeling the Course Management System

- *Logical Schema*
  - Students(sid: *string*, name: *string*, gpa: *float*)
  - Courses(cid: *string*, cname: *string*, credits: *int*)
  - Enrolled(sid: *string*, cid: *string*, grade: *string*)



Corresponding *keys*

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Other Schemata…

- *Physical Schema*: describes data layout
  - Relations as unordered files
  - Some data in sorted order (index)
- *Logical Schema:* Previous slide
- *External Schema*: (Views)
  - Course_info(cid: *string*, enrollment: *integer*)
  - Derived from other tables

Administrators

Applications

Data independence

Concept: Applications do not need to worry about *how the data is structured and stored*

Logical data independence: protection from changes in the *logical structure of the data*

I.e. should not need to ask: can we add a new entity or attribute without rewriting the application?

Physical data independence: protection from *physical layout changes*

I.e. should not need to ask: which disks are the data stored on? Is the data indexed?

One of the most important reasons to use a DBMS

3. Overview of DBMS topics

Key concepts & challenges

Database Management Systems

Database Management System = DBMS

- A collection of files that store the data

Relational DBMS = RDBMS

- Data files are structured as relations (tables)

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...

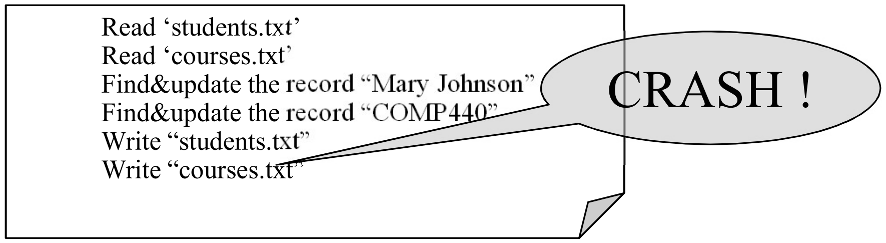
- Enroll “Mary Johnson” in “COMP440”:

Write a C/Java program to do the following:

```
Read ‘students.txt’
Read ‘courses.txt’
Find&update the record “Mary Johnson”
Find&update the record “COMP440”
Write “students.txt”
Write “courses.txt”
```

Problems without an DBMS...

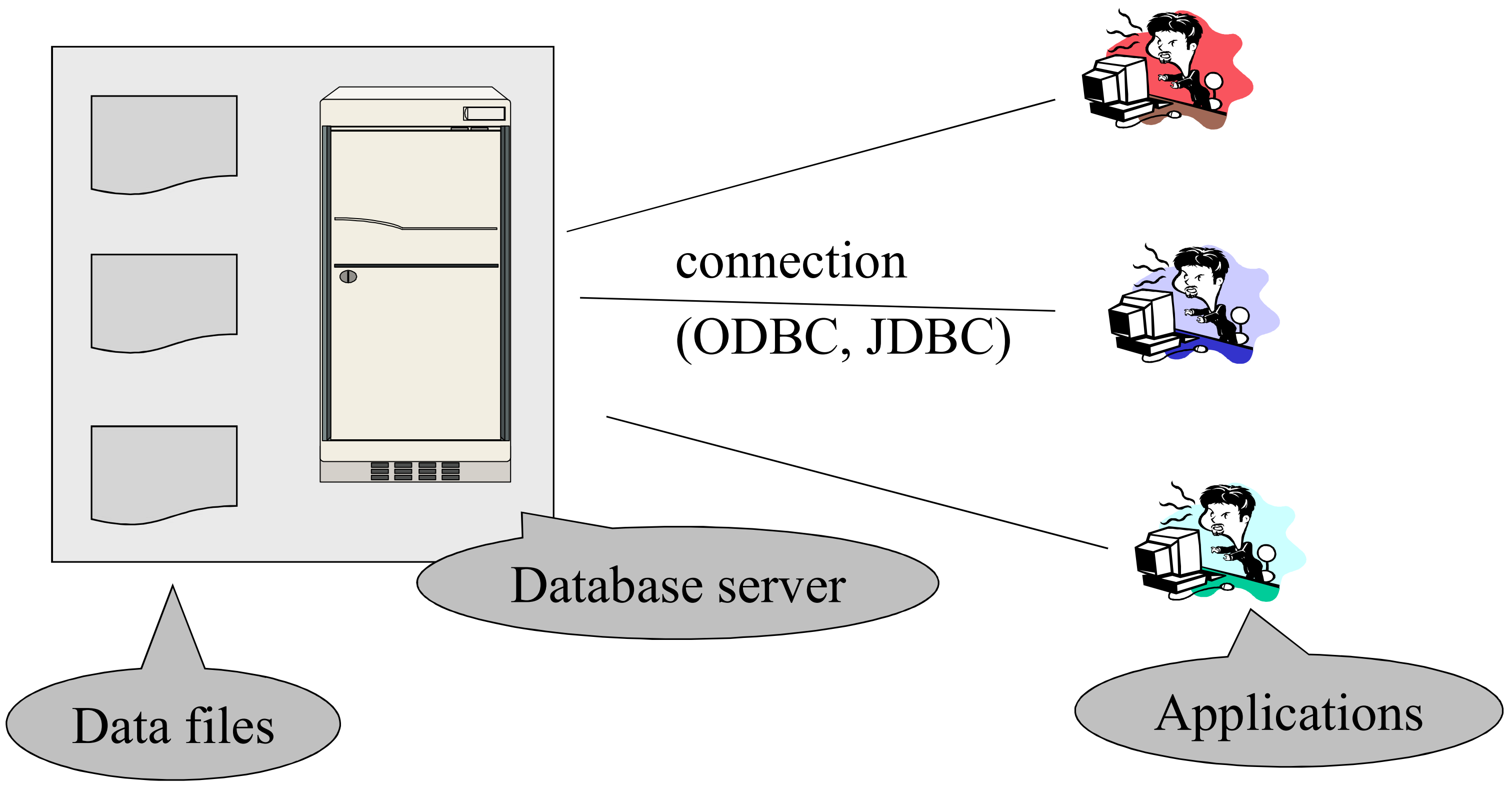
- System crashes:


Read 'students.txt'
Read 'courses.txt'
Find&update the record "Mary Johnson"
Find&update the record "COMP440"
Write "students.txt"
Write "courses.txt"

  - What is the problem ?
- Large data sets (say 50GB)
  - Why is this a problem ?
- Simultaneous access by many users
  - Lock students.txt – what is the problem ?

Enters a DBMS

"Two tier system" or "client-server"



Functionality of a DBMS

The programmer sees SQL, which has two components:

- Data Definition Language - DDL
- Data Manipulation Language - DML
  - query language

Behind the scenes the DBMS has:

- Query engine
- Query optimizer
- Storage management
- Transaction Management (concurrency, recovery)

How the Programmer Sees the DBMS

- Start with DDL to *create tables*:

```
CREATE TABLE Students (
          Name CHAR(30)
          SSN CHAR(9) PRIMARY KEY NOT NULL,
          Category CHAR(20)
)   . . .
```

- Continue with DML to *populate tables:*

```
INSERT INTO Students
VALUES(‘Charles’, ‘123456789’, ‘undergraduate’)
.   .   .   .
```

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
```

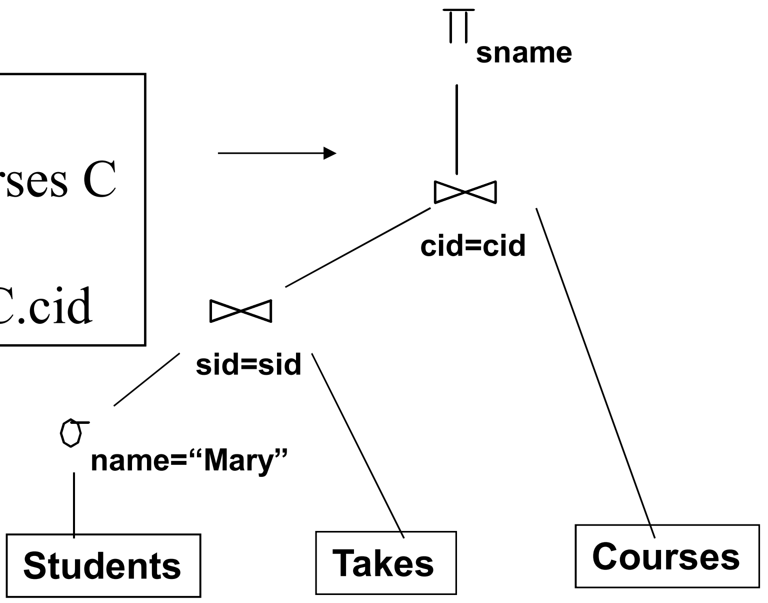
- What happens behind the scene ?
  - Query processor figures out how to answer the query efficiently.

Queries, behind the scene

Declarative SQL query $\longrightarrow$ *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
```

$\longrightarrow$



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

Database Systems

- The big commercial database vendors:
  - Oracle
  - IBM (with DB2)
  - Microsoft (SQL Server)
  - Sybase
- Some free database systems:
  - Postgres
  - MySQL
  - Predator

1. Transactions
2. Concurrency & locking
3. Atomicity & logging
4. Summary

Transactions

- Enroll “Mary Johnson” in “COMP440”:

```
BEGIN TRANSACTION;

INSERT INTO Takes
   SELECT Students.SSN, Courses.CID
   FROM Students, Courses
   WHERE Students.name = ‘Mary Johnson’ and
              Courses.name = ‘COMP440’

-- More updates here....

IF everything-went-OK
     THEN COMMIT;
ELSE ROLLBACK
```

If system crashes, the transaction is still either committed or aborted

Transactions

- A *transaction* = sequence of statements that either all succeed, or all fail
- Transactions have the ACID properties:

  A = atomicity (a transaction should be done or undone completely)

  C = consistency (a transaction should transform a system from one consistent state to another consistent state)

  I = isolation (each transaction should happen independently of other transactions)

  D = durability (completed transactions should remain permanent)

Challenges with Many Users

- Suppose that our CMS application serves 1000’s of users or more- what are some **challenges?**
    - Security: Different users, different roles
    - Performance: Need to provide concurrent access
    - Consistency: Concurrency can lead to update problems

We won’t look at too much in this course, but is extremely important

Disk/SSD access is slow, DBMS hide the latency by doing more CPU work concurrently

DBMS allows user to write programs as if they were the only user

Transactions

- A key concept is the **transaction (TXN)**: an **atomic** sequence of db actions (reads/writes)

Atomicity: An action either completes *entirely* or *not at all*

Acct	Balance
a10	20,000
a20	15,000

Transfer $3k from a10 to a20:

1. Debit $3k from a10
2. Credit $3k to a20

Acct	Balance
a10	17,000
a20	18,000

Written naively, in which states is **atomicity** preserved?

- Crash before 1,
- After 1 but before 2,
- After 2.

DB Always preserves atomicity!

Transactions

- A key concept is the **transaction (TXN)**: an **atomic** sequence of db actions (reads/writes)
    - If a user cancels a TXN, it should be as if nothing happened!

Atomicity: An action either completes *entirely* or *not at all*

- Transactions leave the DB in a **consistent** state
    - Users may write integrity constraints, e.g., ‘each course is assigned to exactly one room’

Consistency: An action results in a state which conforms to all integrity constraints

However**,** note that the DBMS does not understand the *real* meaning of the constraints– consistency burden is still on the user!

Challenge: Scheduling Concurrent Transactions

- The DBMS ensures that the execution of $\{T_1,\ldots,T_n\}$ is equivalent to some **serial** execution
- One way to accomplish this: **Locking**
  - Before reading or writing, transaction requires a lock from DBMS, holds until the end
- **Key Idea***:* If T_i wants to write to an item x and T_j wants to read x, then T_i, T_j **conflict***.* Solution via locking:
  - only one winner gets the lock
  - loser is blocked (waits) until winner finishes

A set of TXNs is isolated if their effect is as if all were executed serially

What if T_i and T_j need X and Y, and T_i asks for X before T_j, and T_j asks for Y before T_i? -> *Deadlock!* One is aborted…

All concurrency issues handled by the DBMS…

Ensuring Atomicity & Durability

- DBMS ensures **atomicity** even if a TXN crashes!
- One way to accomplish this: **Write-ahead logging (WAL)**
- **Key Idea***:* Keep a log of all the writes done.
    - After a crash, the partially executed TXNs are undone using the log

Write-ahead Logging (WAL): Before any action is finalized, a corresponding log entry is forced to disk

We assume that the log is on "stable" storage

All atomicity issues also handled by the DBMS…

Roles in Design, Implementation, and Maintenance of a DBs

- **System Analyst** - specifies system using input from customer; provides complete description of functionality from customer's and user's point of view
- **Data Scientists:** use all available data sources (internal and external) to analyze and gain insights to help decision makers.
- **Database Designer** - specifies structure of data that will be stored in database.

Roles in Design, Implementation and Maintenance of a DBs (con’t)

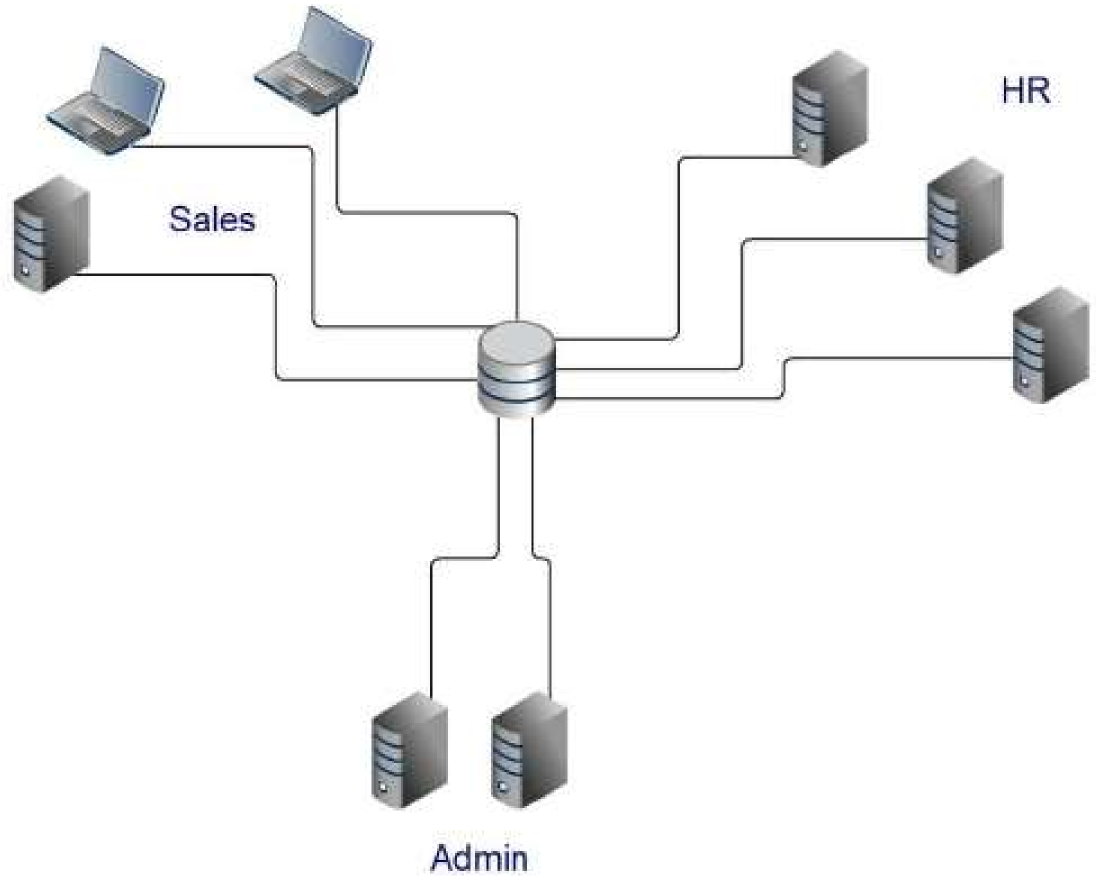
- **Application Programmer** - implements application programs (transactions) that access data and support enterprise rules
- **Database Administrator** - maintains database once system is operational: space allocation, performance optimization, database security
- **System Administrator** - maintains transaction processing system: monitors interconnection of HW and SW modules, deals with failures and congestion

Database Architecture

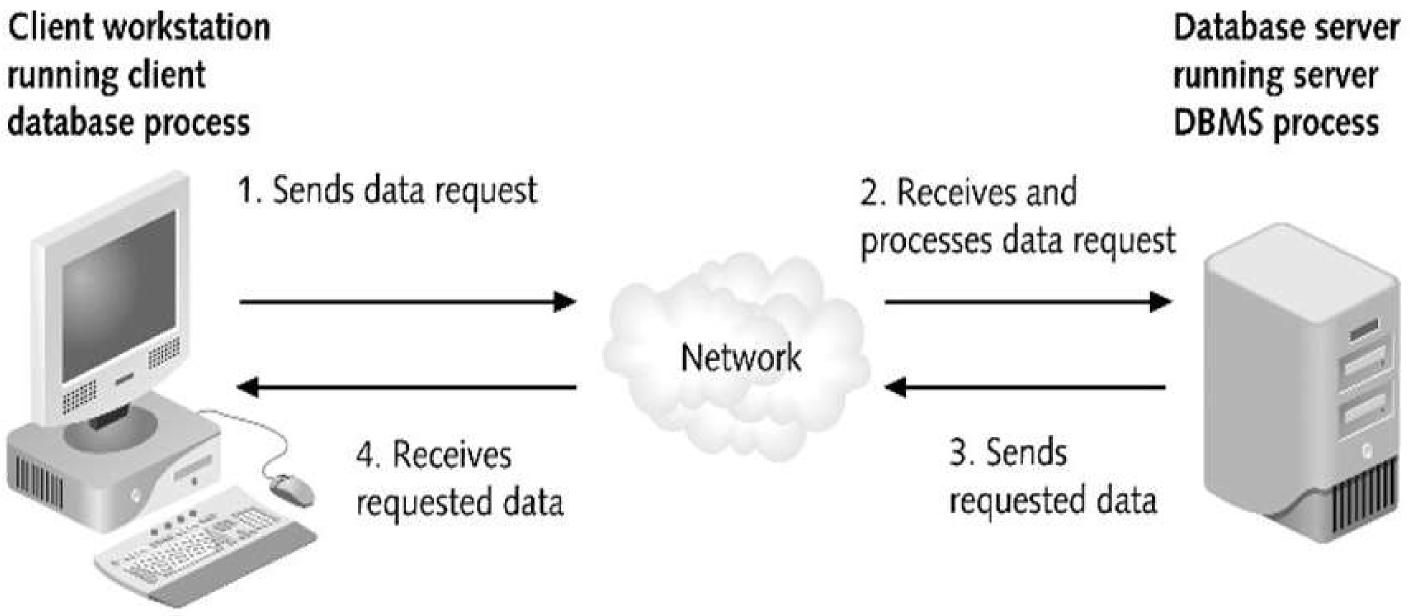
The architecture of a database systems is greatly influenced by
the underlying computer system on which the database is running:

- Database systems structure
  - **Centralized databases**
    - One to a few cores, shared memory
  - **Client-server**
    - One server machine executes work on behalf of multiple client machines.
  - **Parallel databases**
    - Many core shared memory
    - Shared disk
    - Shared nothing
  - **Distributed databases**
    - Geographical distribution
    - Schema/data heterogeneity

Centralized Database



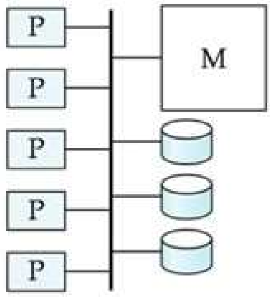
Client/Server Database Architecture


Client workstation running client database process
Database server running server DBMS process
1. Sends data request
2. Receives and processes data request
Network
3. Sends requested data
4. Receives requested data

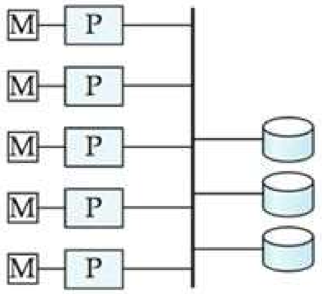
One server machine executes work on behalf of multiple client machines.

Parallel Database Architectures

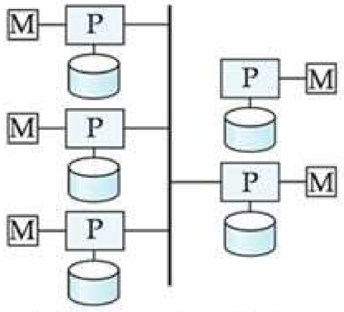
- **Shared memory** -- processors share a common memory
- **Shared disk** -- processors share a common disk, sometimes called clusters
- **Shared nothing** -- processors share neither a common memory nor common disk
- **Hierarchical** -- hybrid of the above architectures



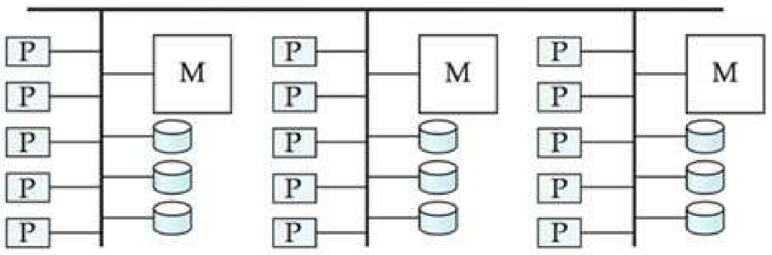
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

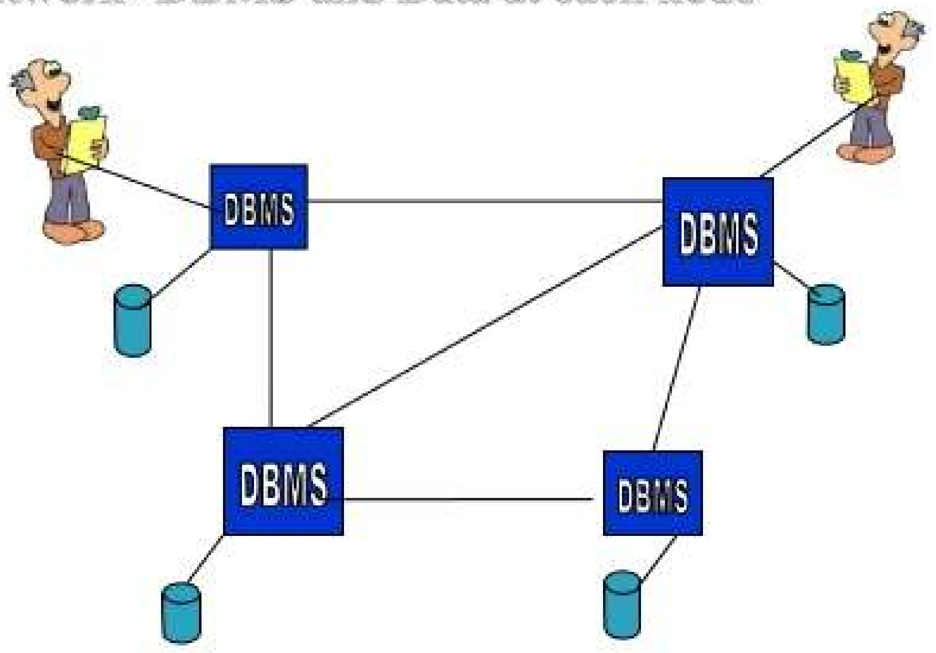
Distributed Database

Geographical distribution

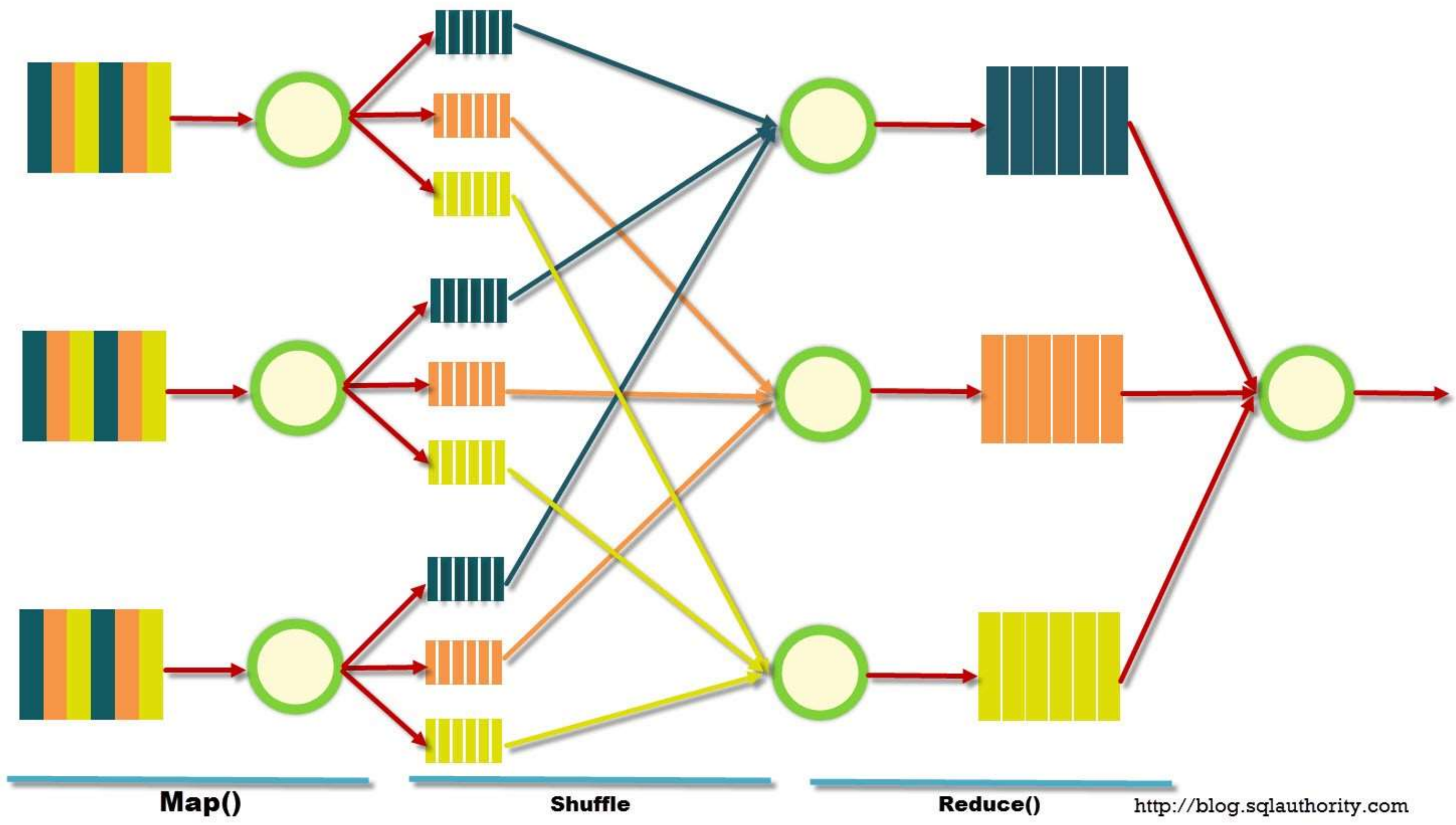
- Communication Network- DBMS and Data at each node

•Users are unaware of the distribution of the data

= Location transparency



How MapReduce Works?



Hadoop - Word Count

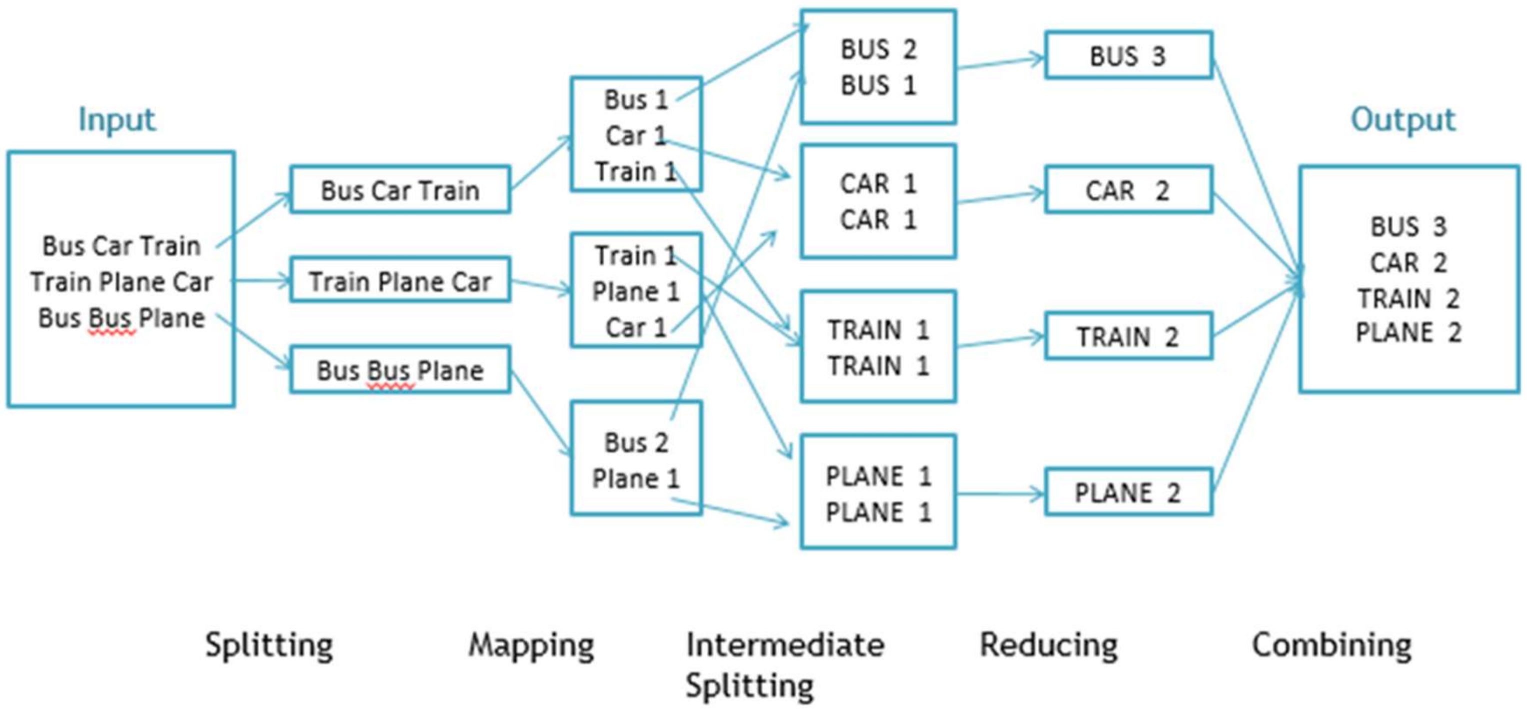
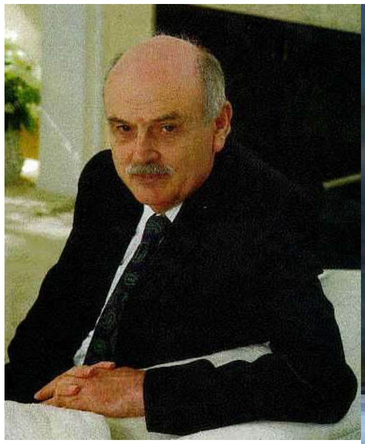


Fig. WorkFlow of MapReducing

Turing Awardees in DB

Charles Bachman (1973)


Edgar F. Codd (1981)

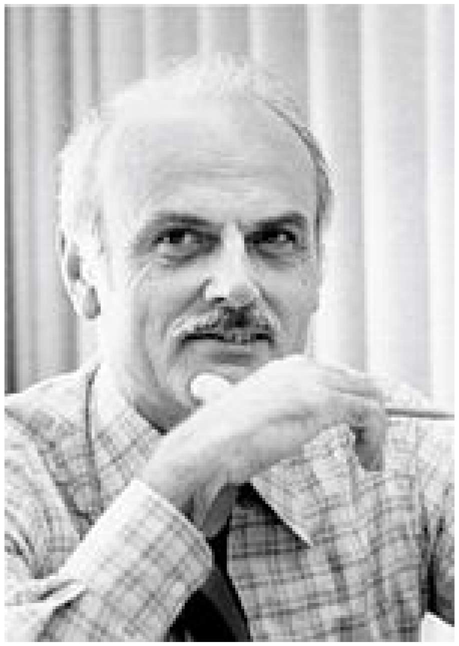
Jim Gray (1998)

Michael Stonebraker (2014)

Charles Bachman

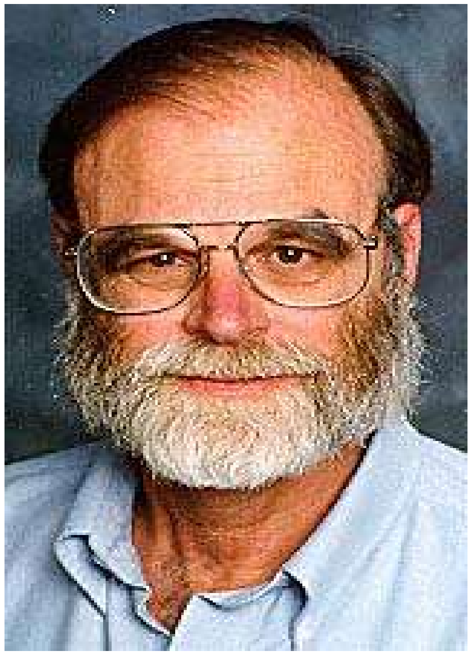
Developer of IDS: the first database system

Edgar. F. Codd



Inventor of the Relational Model

Jim Gray



Founder of Transaction Processing

Michael Stonebraker

Inventor of Ingres and Postgres

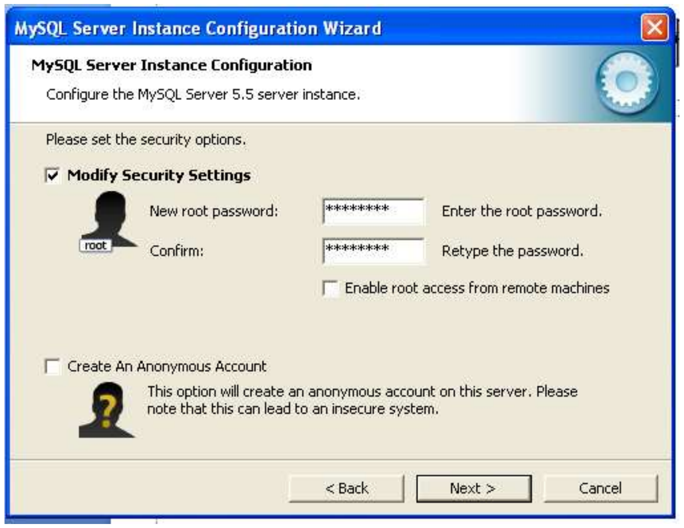
Summary of DBMS

- DBMS are used to maintain, query, and manage large datasets.
    - Provide concurrency, recovery from crashes, quick application development, integrity, and security
- Key abstractions give **data independence**
- DBMS R&D is one of the broadest, most exciting fields in CS. **Fact!**

- Install MySQL

Download Mysql

- www.mysql.com
- Keep the password after setting a new root password, which will be the one you are asked to enter each time Mysql is launched.



MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.5 server instance.

Please set the security options.

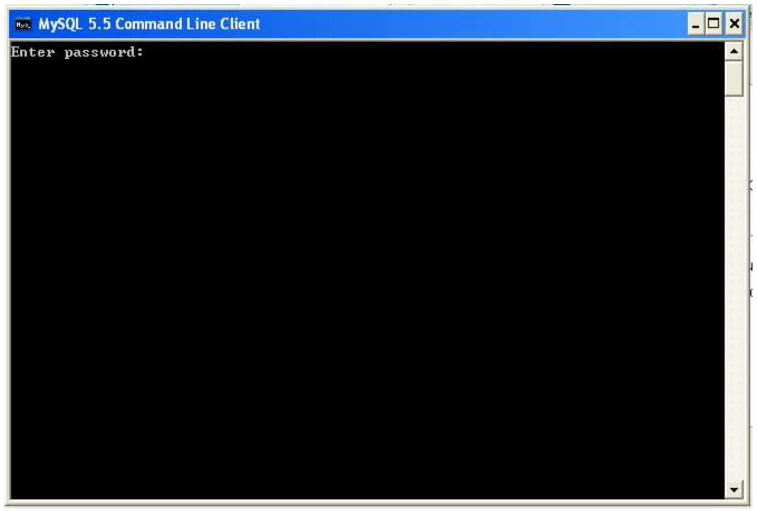
[✓] **Modify Security Settings**

New root password:	*********	Enter the root password.
Confirm:	*********	Retype the password.

[ ] Enable root access from remote machines

[ ] Create An Anonymous Account

This option will create an anonymous account on this server. Please note that this can lead to an insecure system.



MySQL 5.5 Command Line Client

```
Enter password:
```

Download Mysql

- Download the MySQL Community Server 8.0.19 from http://dev.mysql.com/downloads/mysql/

by selecting the right OS platform

- A youtube for installing mysql: https://www.youtube.com/watch?v=iOlJxOkp6sI&spfreload=10. or https://www.youtube.com/watch?v=WuBcTJnIuzo
- Remember the password.
- You can use mysql workbench to manage your connection. http://dev.mysql.com/downloads/workbench/.

Download Mysql for Windows

You can download the windows version from the below website:
https://dev.mysql.com/downloads/installer/.

You can download the .msi file and double click to run the installation process.

Try Example 1

```
show databases;
CREATE database test;
use test;
CREATE TABLE member (
              name varchar(50),
               email varchar(100),
               country varchar(50),
              PRIMARY KEY (name));

INSERT INTO member  VALUES ('Smith‘, ‘smith@gmail.com’, ‘USA’);
SELECT * from member;
```

Try Example 2

```
CREATE TABLE employee (
        empID int(11) NOT NULL  AUTO_INCREMENT,
        firstName varchar(100) DEFAULT NULL,
        PRIMARY KEY (empID));

INSERT INTO employee( firstName)   VALUES ('Smith');
SELECT * from employee;
```

Some Notes

- Always end with ‘;’
- If a column name is repeated, you can get an error message.
- MySQL codes are not case sensitive, but the values in any quotation marks are.