Unit 1
Goals and Informal Synopsis Of The Course
Computer Science is the study of the limits that the laws of physics impose on how information can be efficiently manipulated.

This has deep theoretical/philosophical import in that it gives deep insights into the structure of reality itself.

This also has immense practical import, which has become ever more obvious with computation’s dominance of much of our lives.

In a century of intense study, we’ve managed to isolate a canon of basic concepts that are extremely useful for computer scientists, including for example those of:

• Complexity Theory and Algorithms
• Operating Systems and Networking
• Databases
• Programming Languages and Compilers
• Artificial Intelligence and Machine Learning
Goals Of The Course

- To gain fundamental concepts of state-of-the-art databases (more precisely called: database management systems: DBMSs)
- To get to know key commercial tools used in the design, development, and deployment of database applications
  - Visio for design of relational database
  - Oracle for development of database applications
- To know enough so that it is possible to read/skim a database system manual and
  - Start designing and implementing small databases
  - Start managing, querying, and updating existing
  - Experiment and practice with industrial-quality packages
- Understand the ideas behind recent database-like systems, such as MapReduce, Hadoop, MongoDB
Who am I?

- Adjunct Assistant Professor Eric Mann-Hielscher
- I received my PhD from NYU in 2013, advised by Prof. Shasha
- I also work full time at Google NYC building networking software
- I want to stress: I’m teaching this class because I really want to. I love teaching, and I want to help you reach your individual goals in this course and in your degree as much as I can. Please use me as a resource.

Course Website: [http://cs.nyu.edu/courses/fall15/CSCI-GA.2433-001/](http://cs.nyu.edu/courses/fall15/CSCI-GA.2433-001/)

Course Mailing List: csci_ga_2433_001_fa15@cs.nyu.edu
  - Everyone registered should have been added already

Office hours: 6-7pm on Tuesdays (before class) in CIWW 328

The course is full and has a long waiting list already, so unfortunately if you aren’t already enrolled it’s unlikely you’ll get in
Grading

Grading will break down as follows

• 40%: 6 Homeworks
• 20%: In-Class Midterm Exam
• 40%: Final Exam

I take cheating very seriously, and will report any conduct that doesn’t adhere to the NYU guidelines

• Please just do your own work! You’re here to learn, right?

Homework will involve using Visio and Oracle to generate diagrams and do some hands-on database work. Note that if it’s specified that you must use a particular piece of software for a given Homework, you must use it!

The labs should have this software installed for your use; please make sure you have a CIMS account!

I believe you can get Visio for free from NYU; I’ll make sure and send out instructions to the mailing list
Material Covered

- Methodology used for modeling a business application during the database design process, focusing on entity-relationship models and entity relationship diagrams
- Relational model and implementing an entity relationship diagram in it
- Relational algebra (using SQL syntax)
- SQL as data manipulation language
- SQL as data definition language
- Refining a relational implementation, including the normalization process and the algorithms to achieve normalization
Material Covered

- Physical design of the database using various file organization and indexing techniques for efficient query processing
- Recovery
- Concurrency Control
- Query execution
- Data warehouses
- Online analytical processing (OLAP)
- New systems

Power Point presentations will include more material than will be covered in the course, and will serve as the reference material in place of a textbook
Two Main Functions Of Databases

A very large fraction of computer use is devoted to business processing of data using databases

• Think about what Amazon has to do to manage its operations

Two main uses of databases

• **OLTP** (Online Transaction Processing)
  
The database is used is for entering, modifying, and querying data
  
  Correctness, at least for entering and modifying data must be assured
  
  Example: Amazon charges customer’s credit card for the price of the book that a customer ordered

• **OLAP** (Online Analytical Processing)
  
The database is used for business intelligence, including data mining
  
  The results do not have to be “completely correct,” as this may be too inefficient to guarantee, but complex queries have to be answered (relatively) fast
  
  Example: Amazon wants to know how many books that cost less than $10 each were sold in New Jersey during December 2012
Managing The Data Of An Enterprise

- We may consider some enterprise (organization) and the totality of the information it maintains.
- We think about managing this information, focusing on OLTP.
- Ideally, the information should be stored in a (logically) single (possibly physically distributed) database system.
- We start with a very simple example to introduce some concepts and issues to address.
- We look at only a very small part of information of the type that an enterprise may need to keep.
- We need some way of describing sample data.
- We will think, in this unit, of the database as a set of tables, each stored as a file on a disk.
- It is really a relational database.
Of course, the values do not pretend to be real, they were chosen to be short, so can be easily fit on the slide.

The database talks about employees, books they have checked out from the library (and when), and various illnesses they have had (and when)
Some Typical Queries

Some typical queries

• Give Name of every employee born before 3500
• Give Name and City for every employee who took out a Book after 9000
• Prepare a recall notice to for every employee who had a flu to come for a checkup

Note that some queries involve a single table, and some involve several tables

We would like to have a convenient language, as close as possible to a natural language, to express these queries, and similar ones, thinking of tables, not of lower-level structures (files)

Some languages

• SQL (used to be called Structured Query Language): every relational database supports some “close to standard” version
• QBE (Query By Example); underlying, e.g., Microsoft Access’s GUI
Two Queries in SQL

Imagine that the tables have names (as they of course do in SQL)

- Table1: with columns SSN, City, DOB
- Table2: with columns Name, SSN, DOB, Grade, Salary
- Table3: with columns SSN, Book, Date
- Table4: with columns SSN, Illness, date

Retrieve Name of every employee born before 3500

```
SELECT Name
FROM Table2
WHERE DOB < 3500;
```

Retrieve Grade and City for every employee with the name A

```
SELECT Grade, City
FROM Table2, Table1
WHERE Table2.SSN = Table1.SSN AND Table2.Name = 'A';
```
The Need For Good Design

- It is important also to think carefully about the correct (or simply good enough!) choice of which tables to use and what their structure should be.
- This we should do in order to have good logical design, not worrying (yet) about efficient storage in files.
- Our initial design suffers (for pedagogical reasons) from various problems, which we will see next.
**Redundancy**

A data item appears more than once *unnecessarily*

- Assuming that each SSN has only one DOB, the value of DOB appears twice unnecessarily (in two different tables)
  
  There is a danger that this will be inconsistent

- Even more dangerous would have been duplication of employee’s City
  
  If the employee moves, the City must be changed everywhere it appears

Note, however, that from an efficiency point of view, it might be useful to replicate information, in order to speed up access

- In our example, if we frequently want to correlate DOB with Grade and also DOB with City, it may be good to have it in both tables, and not insist on a “clean” design

Note that *it was necessary* for SSN to appear in two different tables, as otherwise we could not “assemble” information about employees
Storage Of Constraints (Business Rules)

Assume that it is the policy of our enterprise that the value of Salary is determined only by the value of Grade; this is an example of a **business rule** (semantic constraint)

- Thus the fact that the Grade = 2 implies Salary = 80 is written twice in the database
- This is another type of redundancy, which is less obvious at first

There are additional problems with this design.

- We are unable to store the salary structure for a Grade that does not currently exist for any employee.
- For example, we cannot store that Grade = 1 implies Salary = 90
- For example, if employee with SSN = 132 leaves, we forget which Salary should be paid to employee with Grade = 3
- We could perhaps invent a fake employee with such a Grade and such a Salary, but this brings up additional problems, e.g.,

  What is the SSN of such a fake employee?

Note that our constraints specify a pay scale, which is independent of a particular employee
Handling Storage Of Constraints

The problem can be solved by replacing

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>DOB</th>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>121</td>
<td>2367</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>A</td>
<td>132</td>
<td>3678</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>3498</td>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>C</td>
<td>106</td>
<td>2987</td>
<td>2</td>
<td>80</td>
</tr>
</tbody>
</table>

with two tables

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>DOB</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>121</td>
<td>2367</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>132</td>
<td>3678</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>3498</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>106</td>
<td>2987</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
</tr>
</tbody>
</table>
Handling Storage Of Constraints

And now we can store information more naturally

• We can specify that Grade 3 implies Salary 70, even after the only employee with this Grade, i.e., employee with SSN 132 left the enterprise

• We can specify that Grade 1 (a new Grade just established) implies Salary 90, even before any employee with this grade is hired

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>DOB</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>121</td>
<td>2367</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>3498</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>106</td>
<td>2987</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
</tr>
</tbody>
</table>


**Clean Design Versus Efficiency**

However, if the correlation between an employee and salary is needed frequently, e.g., for producing payroll, it may be inefficient to recompute this correlation repeatedly.

So, returning to our original instance of the database, perhaps we should have (despite some redundancy) both the original table and the table associating salaries with grades.

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>DOB</th>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>121</td>
<td>2367</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>A</td>
<td>132</td>
<td>3678</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>3498</td>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>C</td>
<td>106</td>
<td>2987</td>
<td>2</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
</tr>
</tbody>
</table>
One More Problem

- What if it becomes illegal to use social security numbers for anything other than payroll related matters?
- We will have an incredible mess and enormous amount of work to restructure the database, unless we have designed the application appropriately to begin with.
- Of course we did not know that it would become illegal to use social security numbers and it was convenient to do so, so that’s what we used.
- So how to be able to anticipate potential problems?
- NYU had to spend considerable effort to switch from social security numbers to University ID’s.
- We will discuss how to “anticipate” such problems, so such switching is painless.
**Different Users Need Different Data**

- It may be our goal to create a design that best reflects the inherent properties of the data.
  - But, various user groups may need to look at the data assuming different structure (organization) of the data.

- For privacy/security reasons we may want to give different users different access privileges to the database.
  - The payroll department can see salaries but cannot see diseases.
  - The health department can see diseases but cannot see salaries.

- Users may prefer to look at different aspects of the information.
  - The payroll department may prefer to see the salary in a different currency.
  - The health department may prefer to see Age instead of, or in addition to, DOB.
A possible solution: give each user (class of users) privileges to look at a view, that is, a small derived database.

The health department may think that there is a table:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
<th>DOB</th>
<th>Age</th>
<th>Illness</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>121</td>
<td>Portland</td>
<td>2367</td>
<td>47</td>
<td>Flu</td>
<td>2987</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>Boston</td>
<td>3498</td>
<td>25</td>
<td>Cold</td>
<td>3498</td>
</tr>
</tbody>
</table>

The database should provide such a view, which is computed from the existing tables (and the current date), without the user knowing other (prohibited for this user) information.

We need to leave flexibility for unanticipated queries.

- Some people may later be given the right and want to ask the query: “How are salaries and diseases correlated?”
The ideal goal is for the users to both query and modify the database through views.

Unfortunately, sometimes it impossible or difficult to do so:

- If the user wants to change the age of an employee, how should the change be reflected in the date of birth?
  
  There is no unique way of doing it

- How to change the sum of salaries, if some view contains this information?
  
  We want to give a total raise of 5% (increase sum of salaries by 5%), so how to reflect this in individual salaries?
  
  Some employees may get more than 5% and some may get less than 5%
Physical Design

- The database system must be organized so that it is able to process queries efficiently.

- To do this:
  - *Files must be organized appropriately*
  - *Indexes may be employed*

- For example, if we frequently want to find the grade for various SSN, perhaps the file should be hashed on this value, allowing direct access.

- But, if we want to print the salaries of all the employees born in 2783, maybe the file should be sorted by DOB.

- Physical design of databases deals with such issues (including how to distribute information among various sites), which are also closely related to the optimization of query processing.
Recovery

- The database must be resilient even though the system is prone to faults.
- Assume one more table, describing employees' accounts in the credit union

<table>
<thead>
<tr>
<th>SSN</th>
<th>Savings</th>
<th>Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>106</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>121</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>132</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

- We want to give each employee a bonus of 10 in the savings account.
  - To do that, a transaction (execution of a user program) will sequentially change the values of Savings
Example Of A Problem

The file describing the table is stored on a disk, values are read into RAM, modified and written out.

If $X$ is a local variable then we have a trace of the desired execution (in shorthand):

\[
\begin{align*}
X & := \text{Savings}[101] \quad \text{read from disk} \\
X & := X + 10 \quad \text{process in RAM} \\
\text{Savings}[101] & := X \quad \text{write to disk}
\end{align*}
\]

What if the system crashes in the middle, say power goes out?

We do not know which of the values have been changed, so what to do to recover (get back a correct state)?

Various techniques exist for managing the execution, so that reliable execution is possible.
Concurrency

There may also be problems because of the concurrent execution of several transactions in a time sharing system.

Assume that we are running a transaction, T1 (an “reporting” transaction), that should compute and print for each employee the sum of Savings and Checking:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>70</td>
</tr>
<tr>
<td>106</td>
<td>60</td>
</tr>
<tr>
<td>121</td>
<td>80</td>
</tr>
<tr>
<td>132</td>
<td>10</td>
</tr>
</tbody>
</table>

Concurrently SSN = 121 wants to move 40 from Checking to Savings, using transaction T2 (a “moving” transaction).

In a time-sharing system we could have an incorrect execution.

We will write “CH” for Checking and “SA” for Savings.
Execution Trace Of Two Transactions

T1

X1 := SA[121]  (X1 = 0)

X2 := CH[121]  (X2 = 40)
X1 := X1 + X2  (X1 = 40)
PRINT X1  (X1 = 40)

T2

Y1 := CH[121]  (Y1 = 80)
Y1 := Y1 - 40  (Y1 = 40)
CH[121] := Y1

Y2 := SA[121]  (Y2 = 0)
Y2 := Y2 + 40  (Y2 = 40)
SA[121] := Y2

We get 40, an incorrect value of Balance for SSN = 121

Standard operating system constructs do not help here, but concurrency control mechanisms that solve the problem exist in databases (but not in Microsoft Access)
Some Concurrency Aspects

- In the previous examples, we could allow the two transactions to interleave in this way, with the user of the “reporting” transaction being told that correct results are not guaranteed.
- The user may get only an approximate result, which perhaps is sufficient if we are producing “statistical” reports.
- However the database will remain consistent (correct) and the “moving” transaction can execute.
- But if instead of the “reporting” transaction which only read the database, we have a “multiplying” transaction that updates all the values in the database by multiplying them by 2, then the database could be corrupted, and the interleaving cannot be permitted.
The Layers/Levels Of The Ideal Database

- It is customary to think of the database as made of several layers or levels, which are not completely standardized.
- Different levels have different roles.
- We will think of 4 levels:
  - **External** (User) Various user views
  - **Conceptual** (Community) Description of the enterprise
  - **Internal** (Physical) Files, access methods, indexes, distribution
  - **Database O.S.** Recovery and concurrency

- The database, in general, does not run on a bare machine.
- The Database O.S. (DBOS) runs on top of the O.S., such as Windows or Linux.
The Conceptual Level

The conceptual level is most fundamental as it describes the total information and its structure/meaning
• to the extent we understand the information and know how to express our understanding

It is also generally used for manipulating the database, that is querying and modifying it

The tools we have:
• Data Definition Language (DDL), for description
• Data Manipulation Language (DML), for querying and modifying

Tables in our example (their structure, not the specific values which change in time) were a kind of DDL
• They form a schema, a description of the structure.

Of course, this level changes as the needs of the enterprise change
The External Level

- The external level is seen by various users.
- Each view (subschema) is like a small conceptual level.
- It can also change over time.
- A particular view may be modified, deleted, or added even if the conceptual level does not change
  - For example, it may become illegal for some user to see some information
The Internal Level

- The internal level deals with file organization/storage management
- It changes in time too
  - New storage devices are brought
  - Files may have indexes created because some queries have become more frequent
  - The data may be geographically distributed
The Database Operating System Level

- The database operating system level deals with concurrency and recovery
- The database operating system can change too
- The vendor of the database may discover better methods to handle recovery/concurrency
Independence Among Levels

- A very important goal is (Data-) independence between/among levels
- We must make sure that changes in one level disturb the other levels as little as possible (propagate as little as possible)
Who Does What?

The **database vendor/implementer** sends:

- The database operating system
- Tools to create and manipulate the three top levels: external, conceptual, and internal

The **database designers** discuss with the users what information the database should contain and its structure:

- A common model (language for describing reality) is needed for them to communicate.
  
  The *entity-relationship* model is frequently used.

The **database application developers** write the programs (in **SQL** and other languages) that define and manipulate the database.

The **database administrator (DBA)** maintains the database itself (not the specific application programs), including:

- Loading of data, some physical design, backups, tuning, etc.
Challenges

We have seen just the tip of the iceberg of what needs to happen for database systems to function as required.

We need:

- Natural semantics
- Convenient syntax
- Efficiency
- 100% reliability

Enormous effort has been spent since the early 70s to achieve that.
NoSQL: Some Recent Developments

- Not SQL or Not Only SQL
- Need much higher processing throughput
  - In tens of microseconds for stock trading
- Distribute the computations and data accesses among multiple “nodes” to speed up processing
- However building large, fast, correct, and reliable distributed databases is not practical
- So give up “some” correctness and reliability to gain speed
Key Ideas

- Synopsis of the course
- The need for database management systems
- Brief overview of the relational model
- Querying relational database directly and through views
- Need for good logical design
- Need for good physical design
- Recovery
- Concurrency
- Layers of database management systems
- Independence between/among layers
- Various roles of designers, developers, administrators
- NoSQL systems