What are we studying?

- Methods to build databases
  - Data modeling
  - Query languages
  *We’ll try to cover this quickly – many may know it*

- Methods to build DBMSs
  - Storage (safe, persistent)
  - Query (how to make them fast)
  - Transactions (how to make a lot happen at once)
Course Outline
(very rough)

1. Course Introduction
   • Intro / history lesson
   • Relational Model, Data Modeling
2. Relational Database Queries
   • Relational Algebra and SQL
3. Storage mechanisms: Rotating and Otherwise
4. Indexing and Hashing
5. Query Processing
6. Query Optimization
7. Handling Failure
8. Concurrency Control
9. Transaction Management
10. Using a Relational Database
    • Views
    • Constraints
    • Triggers
11. Big Data and Other Advanced Topics

What goes into a DBMS?

- Query Compilation
  - Turn a declarative query to procedural execution
  - What is the fastest way to get the result?
- Transaction Management
  - Try to run lots at once
  - Ensure queries don’t interfere with each other
- Storage Management
  - Disks are slow – how do we get to the data fast?
  - Minimize trips to the disk
Some Goals of a DBMS

- Enhances the accessibility of data, reduces redundancies and inconsistencies
- Simplifies the development of new applications, and the maintenance of existing applications
- Assures data quality, confidentiality, and integrity

Data Models

- A data model allows one to represent real-world entities of interest to a given set of applications
- It is thus useful to identify the basic concepts of such representation; relevant concepts include:
  - **Entity**: an "object" of the application domain
  - **Attribute**: a property of a given entity which meaningful, for the description of the application domain
  
  Each entity is thus characterized by one or more attributes; an attribute takes one or more values, referred to as *attribute values*, from a set of possible values; such set if referred to as *attribute domain*
Data Models: Example

- Consider the example of employees and departments:
  - **Entities**: John Smith, the department #30
  - **Entity sets**: the set of all employees, the set of all departments
  - **Attributes**: employee name, salary, job, department number, department name
  - **Relationships**: the fact that John Smith works in the department #30

The Relational Data Model

- Based on a single data structure – the *relation*
- A relation can be seen as a table
  - rows, called *tuples*
  - columns containing values of specific types, such as integer numbers or strings
### Tables

**representation of relationships**

<table>
<thead>
<tr>
<th>Course-Name</th>
<th>Instructor</th>
<th>Room-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>Smith</td>
<td>DS1</td>
</tr>
<tr>
<td>Operating Syst.</td>
<td>Jones</td>
<td>N3</td>
</tr>
<tr>
<td>Networks</td>
<td>Li</td>
<td>N3</td>
</tr>
<tr>
<td>Security</td>
<td>Li</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Room-Name</th>
<th>Building</th>
<th>Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>Recitation</td>
<td>1</td>
</tr>
<tr>
<td>N3</td>
<td>Recitation</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>Univ. Hall</td>
<td>2</td>
</tr>
</tbody>
</table>

### DBMS: languages

- **Data Definition Language** (DDL). It allows one to define:
  - The logical schema of the DB
  - The semantic integrity constraints
  - The authorizations for data accesses

- **Data Manipulation Language** (DML)
  - Used for data retrieval (query language) and for data updates

- **Storage Definition Language** (SDL)
  - Used to define physical access structures
Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are attributes
- $R = (A_1, A_2, \ldots, A_n)$ is a relation schema
- Example: $instructor = (ID, name, dept\_name, salary)$
- A relation instance $r$ defined over schema $R$ is denoted by $r(R)$.
- The current values a relation are specified by a table
- An element $t$ of relation $r$ is called a tuple and is represented by a row in a table

Attributes

- The set of allowed values for each attribute is called the domain of the attribute
- Attribute values are (normally) required to be atomic; that is, indivisible
- The special value null is a member of every domain. Indicated that the value is “unknown”
- The null value causes complications in the definition of many operations
Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: instructor relation with unordered tuples

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
</tbody>
</table>

Database Schema

- Database schema -- is the logical structure of the database.
- Database instance -- is a snapshot of the data in the database at a given instant in time.
- Example:
  - schema: instructor (ID, name, dept_name, salary)
  - Instance:

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
</tbody>
</table>
Keys

- Let $K \subseteq R$
- $K$ is a **superkey** of $R$ if values for $K$ are sufficient to identify a unique tuple of each possible relation $r(R)$
  - Example: \{ID\} and \{ID, name\} are both superkeys of instructor.
- Superkey $K$ is a **candidate key** if $K$ is minimal
  - Example: \{ID\} is a candidate key for Instructor
- One of the candidate keys is selected to be the **primary key**.
  - Which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - Referencing relation
  - Referenced relation
  - Example: dept_name in instructor is a foreign key from instructor referencing department

Relational Database Design

- Multiple ways to represent data
  - Which is correct?
- Options enforce **constraints**
  - Different sections with different instructors vs. same instructor across all sections
  - Room/time required vs. optional

<table>
<thead>
<tr>
<th>Course</th>
<th>Instructor</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS44800</td>
<td>Clifton</td>
<td>MWF 9:30</td>
<td>PHYS 114</td>
</tr>
<tr>
<td>CS34800</td>
<td>Benotman</td>
<td>Asynchronous</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Instructor</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS44800</td>
<td>Clifton</td>
<td>MWF 9:30</td>
<td>PHYS 114</td>
</tr>
<tr>
<td>CS34800</td>
<td>Benotman</td>
<td>Asynchronous</td>
<td></td>
</tr>
</tbody>
</table>
Relational Database Design

- There is a solid theory behind a good database design
  - Based on the concept of keys and functional dependencies
- **Key**: a given attribute value is unique in the relation
  - E.g., In the courses/rooms/times, each room/time only appears once
- **Functional Dependency**: An attribute value in one relation MUST have a corresponding value in another relation
  
  A good relational database design makes it easy to ensure keys and dependencies hold
- More on this later