Today...

- Relational Data Model - Definitions
  - High level concepts
  - We discuss how these are implemented in SQL
    - More details when we cover SQL
- Front end tools – HTML/CSS/…
- Next class - start with formal query languages before moving to SQL

Database “People” – i.e., roles

- There are several types of database ‘personnel’:
  - Database administrator (DBA) - responsible for installing, maintaining, and configuring the DBMS software.
    - Gets paid the big bucks!
  - Data administrator (DA) - responsible for organizational policies on data creation, security, and planning.
    - Gets to be on the hot seat if data leaks happen.
  - Database designer - defines and implements a schema for a database and associated applications.
    - Logical/Conceptual database designer - interacts with users to determine data requirements, constraints, and business rules.
    - Physical database designer - implements the logical design for a data model on a DBMS. Defines indexes, security, and constraints.
  - DBMS developer - writes the DBMS software code.
  - Application developer - writes code that uses the DBMS.
  - User - uses the database directly or through applications.

Some Terminology

- Database:
  - A collection of related data.
- Data:
  - Known facts that can be recorded and have an implicit meaning.
- Mini-world:
  - Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.
- Database Management System (DBMS):
  - A software package/system to facilitate the creation and maintenance of a computerized database.
    - Example: MySQL, Oracle, MongoDB
- Database System:
  - The DBMS software together with the data and (usually) the applications.
Recent Developments
(Recent in my timeline 😊)

- Social Networks started capturing a lot of information about people and their communications (tweets, photos, videos..)
  - Facebook, Twitter, LinkedIn,....
- All of the above constitutes data
- Search Engines: Google, Bing, Yahoo: collect their own repository of web pages for searching purposes
- New Technologies emerging to manage vast amounts of data generated on the web:
  - Big Data storage systems involving large clusters
  - NOSQL (Not Only SQL) systems
  - A large amount of data now resides on the "cloud" which means it is in huge data centers using thousands of machines.

How will your database system be architected:
Three-Tier Client-Server Architecture

Tier 1: Client (Web/mobile)
- User Interface (using HTML/CSS)

Tier 2: Application Server
- Business logic – written using PHP
- Data processing logic

Tier 3: Database Server
- Data storage/management
- Using MySQL

Data Models..more definitions

- **Data Model:**
  - A formal framework to describe the data and *structure*, constraints, of a database, and
  - the *operations* for manipulating these structures
  - Constructs used to define the database structure
    - Data types, format (table),…ex: name is a char string
- **Constraints:**
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times..ex: GWID is unique

- **Data Model Operations:**
  - These operations are used for specifying database *retrievals* and *updates* by referring to the constructs of the data model.
    - Ex: find student name with GWID= abcd

More terminology..Levels of Data Models

- **Conceptual (high-level, semantic) data models:**
  - Provide concepts that are close to the way many users perceive data.
- **Logical level:**
  - Provide concepts used by DBMS implementations
  - “In earlier definition we mixed conceptual and logical
- **Physical (low-level, internal) data models:**
  - details of how data is stored in the DBMS/computer.
  - These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- **Self-Describing Data Models:**
  - Combine the description of data with the data values. Examples include JSON/XML, key-value stores and some NOSQL systems.
Conceptual Data Model:
The Entity-Relationship (ER) Model

- Provide database design that is easy to interpret by a wide class of users
  - Not just database/CS experts
  - You want to provide a design to a "client" using a representation they can understand
- What’s a natural way to provide an easy to interpret representation…?
  - Fill in the blanks: A ______ is worth a thousand words
- "Visual" representation of the data, how it interacts, constraints, etc.
  - And can be automatically mapped to a data model (relational)
- ER-Model is one such data model
  - We will return to it after we cover the relational model

Example: ER Design for mini-banner:
Entities = Professors, Students, Courses
Relationship between entities

“Who’s taking what, and what grade do they expect?”

One picture provides info on what your system stores and models

UML anyone?

Some more terminology…..
Schemas versus Instances

- Similar to types and variables in programming languages
- Database Schema: structure of the database
  - The description of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
  - Schema Diagram: illustrative display of a database schema.
- Database instance/state: actual data (content) stored in a database at a particular moment in time
  - Initial state: when database is loaded
  - Valid state: A state that satisfies the structures and constraints of the database...
    - job of DBMS to ensure valid entries
  - The database schema changes very infrequently.
    - Preferably never
  - The database state changes every time the database is updated.

Database Schema for a COMPANY Database

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>DEPARTMENT</th>
<th>DEPT_LOCATIONS</th>
<th>PROJECT</th>
<th>WORKS_ON</th>
<th>DEPENDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Mgr</td>
<td>Mgr</td>
<td>Mgr</td>
<td>Mgr_start_date</td>
<td>Mgr</td>
</tr>
<tr>
<td>Dname</td>
<td>Dnumber</td>
<td>Dname</td>
<td>Location</td>
<td>Location</td>
<td>Location</td>
</tr>
<tr>
<td>Phone</td>
<td>Location</td>
<td>Phone</td>
<td>Location</td>
<td>Location</td>
<td>Location</td>
</tr>
<tr>
<td>Hours</td>
<td>Location</td>
<td>Hours</td>
<td>Location</td>
<td>Location</td>
<td>Location</td>
</tr>
<tr>
<td>#</td>
<td>Location</td>
<td>#</td>
<td>Location</td>
<td>Location</td>
<td>Location</td>
</tr>
</tbody>
</table>

| Name | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr |
|----------|----------------|----------------|---------|----------|-----------|
| Name | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr | Mgr |
| Dname | Dnumber | Dname | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location |
| Phone | Location | Phone | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location |
| Hours | Location | Hours | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location |
| # | Location | # | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location | Location |

3
Example of a Database Schema

STUDENT
- Name
- Student_number
- Class
- Major

COURSE
- Course_name
- Course_number
- Credit_hours
- Department

PREREQUISITE
- Course_number
- Prerequisite_number

SECTION
- Section_identifier
- Course_number
- Semester
- Year
- Instructor

GRADE REPORT
- Student_number
- Section_identifier
- Grade

Example Instance & Schema

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>Takes</th>
<th>COURSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
<td>sid</td>
</tr>
<tr>
<td>1</td>
<td>Ross</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Lee</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Emily</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROFESSOR</th>
<th>Teaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>fid</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Wood</td>
</tr>
<tr>
<td>2</td>
<td>Heller</td>
</tr>
<tr>
<td>8</td>
<td>Naraham</td>
</tr>
</tbody>
</table>

- Our focus now: relational schema – set of tables
- Can have other kinds of schemas – XML, object, ...

Populated database state/instance for COMPANY

History of Data Models

- Network Model
- Hierarchical Model
- Relational Model
- Object-oriented Data Models
- Object-Relational Models
- NoSQL/ Big Data technologies
  - Schema-less designs
History of Data Models

- **Network Model:**
  - Network of records
  - The first network DBMS implemented by Honeywell in 1964-65
  - Can model complex relationships between records i.e., graphs!

- **Hierarchical Model:**
  - First implemented by IBM and Rockwell Intl. 1965 (?)
  - Hierarchical network of records (trees like)
  - Abstract view of an Org Chart – models hierarchy in the data

- **Query language:** COBOL
  - Hear about the Y2K bug/problem ?

- **Disadvantages:**
  - Database contains complex set of pointers that thread through sets of records
    - Little scope for “query optimization”, Procedural nature of processing
  - Do you like working with pointers ? 😊

Next: Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation
  - The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations

- start with review of essentials of the formal relational model

- In practice, there is a standard model based on SQL

- **Note:** There are several important differences between the formal model and the practical model
  - And, there are variations in SQL features provided on different DBMS systems
    - Oracle SQL, MySQL, MS-SQL, ...

Relational Model Concepts

- The model was introduced by Dr. E.F. Codd of IBM Research in 1970
  - "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970
  - Describes the data minimally and mathematically
    - A relation describes an association between data items – tuples with attributes
    - Uses standard mathematical (logical) operations over the data – relational algebra or relational calculus

- It contributed: data independence, query languages, query optimization

- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

Why Did It Take So Many Years to Implement Relational Databases?

- Codd’s original work: 1969-70
- Earliest relational database research: ~1976
- Commercial Relational DBMSs: ~mid 1980s
- Widespread deployment” mid-1990’s
- Why the gap? Top 10 reasons…
  1. "You could do the same thing in other ways"
  2. "Nobody wants to write math formulas"
  3. "Why would I turn my data into tables?"
  4. "It won’t perform well"
  5. …

- **What do you think?**
Relational Model Definitions

- A **relation** is a table with columns and rows.
- An **attribute** is a named column of a relation.
- A **tuple** is a row of a relation.
- A **domain** is a set of allowable values for one or more attributes.
- The **degree** of a relation is the number of attributes it contains.
- The **cardinality** of a relation is the number of tuples it contains.
- A **relational database** is a collection of normalized relations with distinct relation names.

Example of a Relation

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Set</th>
<th>Home_phone</th>
<th>Address</th>
<th>Office_phone</th>
<th>Age</th>
<th>Spk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benjamin</td>
<td>305-61-2435</td>
<td>973-1816</td>
<td>2918 Bluebonnet Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.21</td>
</tr>
<tr>
<td>Chung-Fla</td>
<td>381-62-1245</td>
<td>375-4409</td>
<td>125 Kirby Road</td>
<td>NULL</td>
<td>18</td>
<td>2.89</td>
</tr>
<tr>
<td>Dick David</td>
<td>422-11-2300</td>
<td>3452 Eiger Road</td>
<td>NULL</td>
<td>749-1283</td>
<td>28</td>
<td>3.03</td>
</tr>
<tr>
<td>Rohan</td>
<td>489-22-1100</td>
<td>978-9821</td>
<td>125 Lark Lane</td>
<td>749-8462</td>
<td>28</td>
<td>3.03</td>
</tr>
<tr>
<td>Barbara</td>
<td>533-69-1238</td>
<td>839-8461</td>
<td>7284 Fontana Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.25</td>
</tr>
</tbody>
</table>
```

Degree= 7, Cardinality = 5

Relation Schemas and Instances

- A **relation schema** is a definition of a single relation.
- A **database schema** is a set of relation schemas (modeling a particular domain).

- A **relation instance** denoted **r(R)** over a relation schema R(A₁, A₂, ..., Aₙ) is subset of the Cartesian product of the domains of all attributes in the relation schema
  - **r(R)** ⊆ dom(A₁) × dom(A₂) × ... × dom(Aₙ)
  - i.e., a set of n-tuples <d₁, d₂, ..., dₙ> where each dᵢ is an element of dom(Aᵢ) or is null.
  - A value of null represents a missing or unknown value.
Definition Summary

<table>
<thead>
<tr>
<th>Informal Terms</th>
<th>Formal Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Relation</td>
</tr>
<tr>
<td>Column Header</td>
<td>Attribute</td>
</tr>
<tr>
<td>All possible Column</td>
<td>Domain</td>
</tr>
<tr>
<td>Values</td>
<td>Tuple</td>
</tr>
<tr>
<td>Row</td>
<td>Number of rows</td>
</tr>
<tr>
<td>Cardinality</td>
<td></td>
</tr>
<tr>
<td>Table Definition</td>
<td>Schema of a Relation</td>
</tr>
<tr>
<td>Populated Table</td>
<td>State of the Relation</td>
</tr>
</tbody>
</table>

Properties of relations

- Each relation name is unique
  - No two relations have the same name
- Each cell of the relation (value of a domain) contains exactly one atomic (single) value
- Each attribute of a relation has a distinct name
- Values of an attribute are all from the same domain
- Each tuple is distinct. There are no duplicate tuples
  - Properties of a relation; in SQL can be bags (allow duplicates)
  - Order of attributes is not really important
- Note difference from mathematical def of relations
  - Tuple \((x,y)\) is not the same as \((y,x)\) in def of relation
  - Reason: attribute names represent domain and can be reordered
- Order of tuples is not important

Properties and constraints of the data

- Data represented as a relation/table
- Schema specifies the attributes/columns and their type

Any other properties we need to define to capture the “application”

Need to capture the ‘business’ rules:
  - How do we uniquely identify a student?
  - Can their letter grade be any alphabet?
  - Can a student take a course that is not in the course schedule/bulletin?
  - ...

Concept of Constraints

CONSTRAINTS

Constraints determine which values are permissible (or not) in the database. Three main types:

- **Inherent or Implicit Constraints:**
  - These are based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
- **Schema-based or Explicit Constraints – Integrity Constraints:** are rules or restrictions that apply to the database and limit the data values it may store
  - They are expressed in the schema by using the facilities provided by the model.
- **Application based or semantic constraints:**
  - These are beyond the expressive power of the model and must be specified and enforced by the application programs.
Integrity Constraints (ICs)
- **IC**: condition that must be true for any instance of the database; e.g., domain constraints.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A legal instance of a relation is one that satisfies all specified ICs.
- Why is this useful
  - If the DBMS checks ICs, stored data is more faithful to real-world meaning.
- **Think of the constraints as the business rules derived from the application**
- Carefully analyze the semantics of the real world app before defining the constraints

Where do ICs Come From?
- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
  - Need to carefully analyze the application before reaching a conclusion on the Integrity Constraints!
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.
  - Shall return to these after we cover DML aspect of SQL

Definition: Relational Model Integrity Constraints
- Integrity rules are used to ensure the data is accurate.
- Types of constraints:
  - **Domain constraint**: Every value for an attribute must be an element of the attribute's domain or be null.
    - null represents a value that is currently unknown or not applicable.
    - null is not the same as zero or an empty string.
  - **Entity integrity constraint**: In a base relation, no attribute of a primary key can be null.
  - **Key constraint**: every relation must have a key
  - **Referential integrity constraint**: If a foreign key exists in a relation, then the foreign key value must match a primary key value of a tuple in the referenced relation or be null.

Key Constraints
- **Superkey** of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1(SK) = t2(SK)
    - This condition must hold in any valid state r(R)
- **Candidate Key** of R:
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)
  - A Key is a Superkey but not vice versa
- **Primary Key**: a DBA chosen key for the relation/table
Primary Key Constraints

- Every relation must have a key
- A set of fields is a key for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
- Part 2 false? A superkey.
- If there’s >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key.
- E.g., what is a key for Students relation?
- sid is a key for Students. (What about name?) The set \{sid, gpa\} is a superkey.

Key Constraints (continued)

- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - CAR has two keys:
    - Key1 = \{State, Reg#\}
    - Key2 = \{SerialNo\}
  - Both are also superkeys of CAR
  - \{SerialNo, Make\} is a superkey but not a key.
- In general:
  - Any key is a superkey (but not vice versa)
  - Any set of attributes that includes a key is a superkey
  - A minimal superkey is also a key
  - Can have many candidate keys, one of them chosen as primary key

Key Constraints (continued)

- If a relation has several candidate keys, one is chosen arbitrarily to be the primary key.
  - The primary key attributes are underlined.
- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - We chose SerialNo as the primary key
  - The primary key value is used to uniquely identify each tuple in a relation
  - Provides the tuple identity
  - Also used to reference the tuple from another tuple
- General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
- Not always applicable – choice is sometimes subjective

Referential Integrity

- A constraint involving two relations
  - The previous constraints involve a single relation.
- Used to specify a relationship among tuples in two relations:
  - The referencing relation and the referenced relation.
- Examples:
  - Student enrolled in a course
  - Employee working on a project: in Works_On table
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.
  
  - If a value of sid appears in Enrolled relation then it MUST appear in Students relation
    “Only students can take courses”

<table>
<thead>
<tr>
<th>Enrolled</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
</tr>
</tbody>
</table>

Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields in one relation that is used to ‘refer’ to a tuple in another relation. (Must correspond to primary key of the second relation.)
  
  - Like a 'logical pointer'.
  
  - In Enrolled table – sid is a student, what can we say about the students table?
    - Enrolled(sid: string, cid: string, grade: string)

  - sid in enrolled is a foreign key referring to Students:
    - The student with this ID MUST exist in the Students table
    - If all foreign key constraints are enforced, referential integrity is achieved, i.e., no dangling references.

Referential Integrity (or foreign key) Constraint – more general definition

- Statement of the constraint
  
  - The value in the foreign key column (or columns) FK of the referencing relation R1 can be either:
    1. a value of an existing primary key value of a corresponding primary key PK in the referenced relation R2, or
    2. a null.
  
  - In case (2), the FK in R1 should **not** be a part of its own primary key.

Other Types of Constraints

- Semantic Integrity Constraints:
  
  - based on application semantics and cannot be expressed by the model per se
  
  - Example:
    - “the max. no. of hours per employee for all projects he or she works on is 40 hrs per week”;
    - Grade cannot be any alphabet; …
  
  - A constraint specification language may have to be used to express these
  
  - SQL-99 allows CREATE TRIGGER and CREATE ASSERTION to express some of these semantic constraints
  
  - Keys, Permissibility of Null values, Candidate Keys (Unique in SQL), Foreign Keys, Referential Integrity etc. are expressed by the CREATE TABLE statement in SQL.
When do Integrity Constraints get triggered…

**Populated database state**
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide shows an example state for the COMPANY database schema
- The update operations must keep the database in a consistent state – i.e. all instances must satisfy integrity constraints
- Updates may **propagate** to cause other updates automatically. This may be necessary to maintain integrity constraints.

---

**COMPANY Database Schema**

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>Name</th>
<th>Mini</th>
<th>Lname</th>
<th>Stnl</th>
<th>Bdte</th>
<th>Address</th>
<th>Sex</th>
<th>Salary</th>
<th>Super</th>
<th>Dno</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>Dname</th>
<th>Dnumber</th>
<th>Mgr_ssn</th>
<th>Mgr_start_date</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DEPT_LOCATIONS</th>
<th>Dnumber</th>
<th>Location</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>Projname</th>
<th>Projnumber</th>
<th>Plocation</th>
<th>Drum</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th>Ensn</th>
<th>Proj</th>
<th>Hours</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DEPENDENT</th>
<th>Ensn</th>
<th>Dependent_name</th>
<th>Sex</th>
<th>Bdte</th>
<th>Relationship</th>
</tr>
</thead>
</table>

---

**Figure 5.5**
Schema diagram for the COMPANY relational database schema.
Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

Possible violations for each operation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

Possible violations for each operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 6 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

Possible violations for each operation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
  - Any of the other constraints may also be violated, depending on the attribute being updated:
    - Updating the primary key (PK):
      - Similar to a DELETE followed by an INSERT
      - Need to specify similar options to DELETE
    - Updating a foreign key (FK):
      - May violate referential integrity
    - Updating an ordinary attribute (neither PK nor FK):
      - Can only violate domain constraints
Relational Query Languages
- A major strength of the relational model: supports simple, powerful querying of data.
  - Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Query languages:
- Allow specification of schemas and constraints
- Allow manipulation and retrieval of data from a database.

Relational model supports simple, powerful QLs:
- Strong formal foundation based on logic.
- Allows for much optimization.

Query Languages ≠ programming languages!
- QLs not expected to be “Turing complete”.
- QLs not intended to be used for complex calculations.
- QLs support easy, efficient access to large data sets.

Formal Query Languages
- Formal query languages are defined as mathematical operators over the set
  - advantage of a formal language?
  - Relational algebra, Relational calculus are examples

Procedural vs Non-procedural languages
- Procedural: what data to fetch from DB and how/where to get the data
- Non-procedural: what data to fetch from DB
  - System/DBMS needs to figure out the “how”
- Can have a mix in practice
  - Relational algebra: procedural language
  - Relational calculus: non-procedural (declarative)

SQL: Structured Query Language
The standard language for relational data
- Invented by folks at IBM, esp. Don Chamberlin
- Actually not a great language...
- Beat a more elegant competing standard, QUEL, from Berkeley

Separated into a DML & DDL
SQL DML component based on relational algebra & calculus
SQL
- components
  - Data definition (DDL) – to define schema/tables
    Define Schema
  - Define Constraints
  - Manipulation/query (DML) – for queries
  - Transaction control – to specify a transaction
  - Index – to specify storage and indexing schemes
  - Authorization – for access control/security
  
  We will cover the DDL and query part of SQL first
  Shall return to the other components after we cover those topics

The Big Picture: SQL to Algebra to Query Plan to Web Page

```
SELECT *
FROM STUDENT, Takes, COURSE
WHERE STUDENT.sid = Takes.sID
AND Takes.cID = cid
```