

CS 178: Database Management Systems

Entity-Relationship (ER) Model
Intro to Relational Model

DBMS...

A little history

- ❖ In DBMS: single instance of data maintained and accessed by different users
- ❖ File Processing: Earliest form of information storage systems
- ❖ each user keeps files needed for specific application
 - one user keeps track of students fees and payments
 - second user keeps files on student grades

Progression of Database Systems

- ❖ Early 1950's- file proc, IBM's Rmac system
- ❖ 1960's: first generalized DBMS IBM Sabre
- ❖ 1970's: Relational model proposed, INGRES, System R, Query languages- Sequel(SQL),QUEL
- ❖ 1980's: DBMS for PCs, commercial RDBMS- Oracle, Sybase, Informix
- ❖ 1990s: Object-relational DBMS, Multimedia, Spatial/GIS, Data Mining/OLAP, Dist.DB

Part 1: Schedule

- ❖ Start with Data Models
 - Relational Model with a little ER model intro
- ❖ Formal query languages- Relational algebra
- ❖ SQL
- ❖ Database schema design: how to design a “good” schema, how to measure “good”?
 - Normal Forms (3NF, BCNF)
- ❖ Demonstrate concepts learnt on Commercial DBMS – Oracle, MySQL

How Does One Build a Database?

- ❖ Requirements Analysis: what data, apps, critical operations
- ❖ Start with a conceptual **model**
- ❖ Design & implement **schema**
- ❖ Write **applications** using DBMS and other tools
 - Many ways of doing this where the hard problems are taken care of by other people (DBMS, API writers, library authors, web server, etc.)
 - Common applications include PHP/JSP/servlet-driven web sites
- ❖ The DBMS takes care of query **optimization** and execution

Getting More Concrete: Building a Database and Application

1. Start with a conceptual model
 - “On paper” using certain techniques
 - E-R Model
 - ignore low-level details – focus on logical representation
 - “step-wise refinement” of design with client input
2. Design & implement **schema**
 - Design and codify (in SQL) the relations/ tables
 - Refine the schema – *normalization*
 - Do **physical** layout – indexes, etc.
3. Import the data
4. Write applications using DBMS and other tools
 - Many of the hard problems are taken care of by other people (DBMS, API writers, library authors, web server, etc.)

Why use a graphical language ?

- ❖ Convey database design and properties in simple but precise manner
- ❖ Interpreted by any type of user
 - Does not need to know anything about CS
- ❖ Capture the business rules of the application

- ❖ Picture is worth a thousand words

Graphical Language for Describing Systems

- ❖ Example ????
- ❖ Where did you first see such an example ?
 - Describe your programs using ????

An Example: "mini" banner

- ❖ Database containing information about
 - Students
 - Faculty
 - Courses
- ❖ Students take courses
- ❖ Faculty teach courses
- ❖ How to 'define' student/faculty/course ?

Designing a Schema (Set of Relations)

STUDENT

sid	name
1	Jill
2	Bo
3	Maya

Takes

sid	cid
1	550-0103
1	700-1003
3	500-0103

COURSE

cid	name	sem
550-0103	DB	F03
700-1003	AI	S03
501-0103	Arch	F03

PROFESSOR

fid	name
1	Ives
2	Saul
8	Roth

Teaches

fid	cid
1	550-0103
2	700-1003
8	501-0103

Entity Relationship Model

- ❖ Based on collection of real world objects or concept called *entities*; ex: employee, student
 - *attribute* represents properties of entity; s.s.num
- ❖ *relationship* represents interaction between entities
- ❖ overall logical structure represented by ER diagram representing entity sets, relationships, attributes

ER Model Basics

- ❖ Conceptual design:
 - What are the *entities* and *relationships* in the enterprise?
 - What information about these entities and relationships should we store in the database?
 - What are the *integrity constraints* or *business rules* that hold?
 - Can map an ER diagram into a relational schema.






ER Model Basics

- ❖ Entity: Real-world object distinguishable from other objects.
 - An entity is described (in DB) using a set of attributes.
- ❖ Entity Set: A collection of similar entities. E.g., all employees.
 - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
 - Each entity set has a *key*.
 - Each attribute has a *domain*.
- ❖ Representation:
 - Entity set represented by rectangle
 - Attribute represented by Oval
 - Key attribute underlined

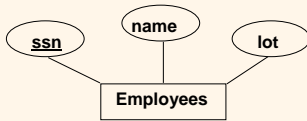
ER Model Basics (Contd.)

- ❖ Relationship: Association among two or more entities. E.g., Dan takes Database Course; Attishoo works in Pharmacy department.
 - Represented by a Diamond symbol
- ❖ Relationship Set: Collection of similar relationships.
 - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 E1, ..., en En
 - Same entity set could participate in different relationship sets, or in different "roles" in same set.

Conceptual Design Process

- ❖ What are the entities being represented? 
- ❖ What are the relationships? 
- ❖ What info (attributes) do we store about each? 

- ❖ What keys & integrity constraints do we have? 

ER Model Basics



... OK, But What about Connectivity in the E-R Diagram?

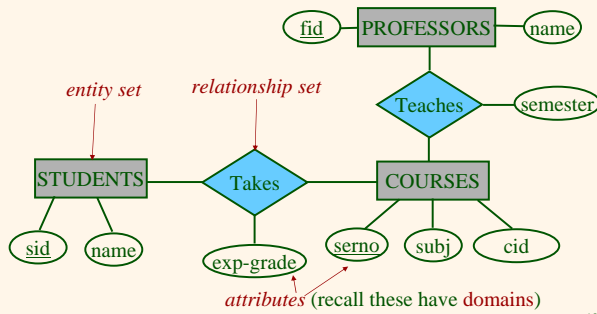
- ❖ Attributes can *only* be connected to entities or relationships
- ❖ Entities can *only* be connected via relationships
- ❖ As for the edges, let's consider kinds of relationships and integrity constraints...



(warning: the book has a slightly different notation here!)

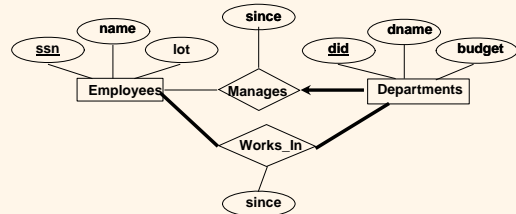
Entity-Relationship Diagram for the Example

Underlined attributes are keys



Another Example: A Company Database

- ❖ Entities: Employees, Departments
 - Employees work in a department
 - Each department has a manager

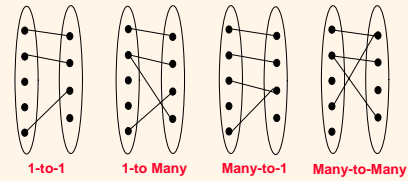


Constraints – Key and Participation

- ❖ Capture properties of the relationship and entities
- ❖ Every entity set has a key attribute
 - Not two elements can have the same value on this attribute
- ❖ Does every element in the entity set appear/participate in the relationship ?
- ❖ Define constraints based on properties of the mapping/ relation between entity sets

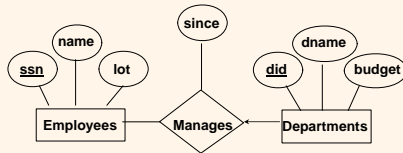
Properties of relations

- ❖ Binary relationships can be classified as one-to-one, many-to-one, one-to-many, many-to-many
- ❖ What is the type of mapping/ relation



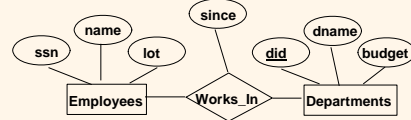
Manager Relationship

Want to model the info that a department has one manager:
Type of mapping ???



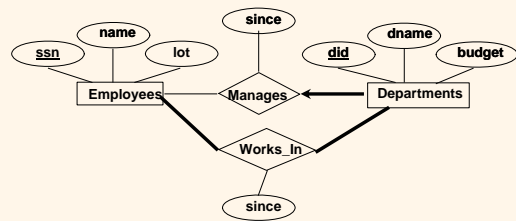
Works-in Relationship

- ❖ Want to model the info that every employee is assigned to a department:
Type of mapping ??



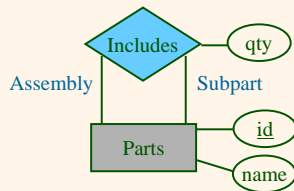
- ❖ Consider Works_In: An employee can work in many departments; a dept can have many employees.
 - So what type of mapping ?
- ❖ In contrast, each dept has at most one manager, according to the *key constraint* on Manages.
- ❖ Does every department have a manager?
 - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total (vs. partial)*.
 - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)
- ❖ Is every employee a manager ?

Overall ER-Diagram for the two relations



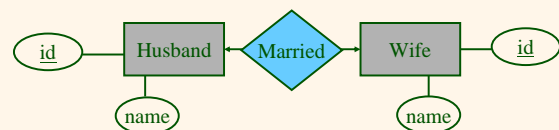
Roles: Labeled Edges

Sometimes a relationship connects the same entity, and the entity has more than one *role*:

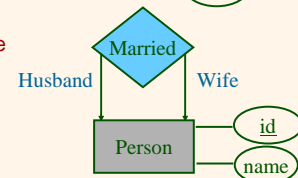


This often indicates the need for *recursive queries*

Roles vs. Separate Entities

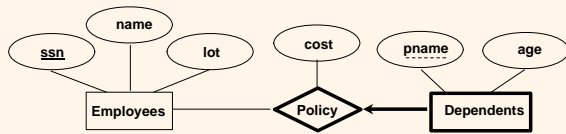


What is the difference between these two representations?



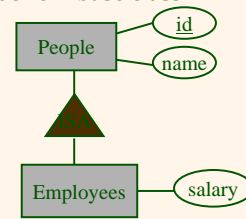
Weak Entities

- ❖ A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
 - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
 - Weak entity set must have total participation in this *identifying* relationship set.
 - If Employee is deleted, then we **MUST** delete the dependent



ISA Relationships: Subclasses (Structurally)

- ❖ Inheritance states that one entity is a “special kind” of another entity: “subclass” should be member of “base class”



Relationships: Binary or n-ary

- ❖ Binary: Relationship between two entity sets
- ❖ N-ary: Relationship between any N entity sets
 - Not all n-ary can be converted to a set of binary relationships

Conceptual Design Using the ER Model

- ❖ Design choices:
 - Should a concept be modeled as an entity or an attribute?
 - Should a concept be modeled as an entity or a relationship?
 - Identifying relationships: constraints, type, participation
- ❖ Constraints in the ER Model:
 - A lot of data semantics can (and should) be captured.
 - But some constraints cannot be captured in ER diagrams.

Summary of Conceptual Design

- ❖ *Conceptual design follows requirements analysis,*
 - Yields a high-level description of data to be stored
 - Visual language - the diagram is the syntax!
- ❖ ER model popular for conceptual design
 - Constructs are expressive, close to the way people think about their applications.
 - There are additional constructs in a “real” ER model based tools.
- ❖ Can automate mapping of ER model to relational tables!

The Relational Database Model

The Relational Data Model (1970)

Originally proposed by E.F. Codd (IBM):

- Separates physical implementation from logical
- Models the data **independently** from how it will be used (accessed, printed, etc.)
 - 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**

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

- ❖ Codd's original work: 1969-70
- ❖ Earliest relational database research: ~1976
- ❖ Why the gap?
 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”

❖ *What do you think?*

Relational Model

- ❖ Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- ❖ Shift from networked “records” of data to a set of tables
- ❖ Row corresponds to relationship among values
- ❖ Table corresponds to a relation
 - mathematical definition of relation over K sets

Relational Data Model: Definitions

- ❖ *Relational database*: a set of *relations* or *tables*.
 - Columns of a relation are called *attributes* or *fields*
 - The number of these columns is the *arity* of the relation
 - The rows of a relation are called *tuples*
 - The number of rows is the *cardinality/size*
 - Each attribute has values taken from a *domain*, e.g., name has domain *string*

Relational Model: Definition

- ❖ Formally, a table is a relation over K sets (domains)
 - $R = D_1 \times D_2 \times \dots \times D_K$
 - Tuple = (t_1, t_2, \dots, t_k) , where t_i is an element from domain/set D_i
- ❖ A database is a collection of relations
- ❖ Theoretically: a relation is a *set* of tuples; no tuple can occur more than once
 - *Real systems may allow duplicates for efficiency or other reasons – we’ll ignore this for now*

Integrity Constraints

- ❖ Domains and schemas are one form of constraint on a valid data instance
- ❖ Other important constraints include:
 - Key constraints:*
 - Subset of fields that *uniquely identifies* a tuple, and for which no subset of the key has this property
 - May have several *candidate* keys; one is chosen as the *primary* key
 - A *superkey* is a subset of fields that includes a key
 - Inclusion dependencies* (referential integrity constraints):
 - A field in one relation may refer to a tuple in another relation by including its key
 - The referenced tuple must exist in the other relation for the database instance to be valid

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.

Formal Query Languages

- ❖ Formal query languages are defined as mathematical operators over the set
 - What is the advantage of a formal language ?
 - Relational algebra, Relational calculus are examples
- ❖ Procedural vs Non-procedural languages
 - Can have a mix in practice

The SQL Query Language

- ❖ The standard language for relational databases
 - Developed by IBM (system R) in the 1970s
- ❖ Based on relational algebra and relational calculus
- ❖ Many components to SQL
- ❖ Standards:
 - SQL-86
 - SQL-89 (minor revision)
 - SQL-92 (major revision)
 - SQL-99 (major extensions, current standard)

SQL

- ❖ Has many components
 - Data definition (**DDL**) – to define tables
 - 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

Creating Relations in SQL

- ❖ Creates the Students relation. Observe that the type (**domain**) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
(sid: CHAR(20),
 name: CHAR(20),
 login: CHAR(10),
 age: INTEGER,
 gpa: REAL)
```

Creating Relations in SQL

- ❖ Creates the Students relation. Observe that the type (**domain**) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
(sid: CHAR(20),
 name: CHAR(20),
 login: CHAR(10),
 age: INTEGER,
 gpa: REAL)
```

- ❖ As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
(sid: CHAR(20),
 cid: CHAR(20),
 grade: CHAR(2))
```

- Is sid same field in the two tables??

More on 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.
 - DBMS should not allow illegal instances.
- ❖ Why is this useful
 - If the DBMS checks ICs, stored data is more faithful to real-world meaning.
 - Avoids data entry errors, too!
- ❖ Think of the constraints as the business rules derived from the application

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.

Primary and Candidate Keys in SQL

❖ Possibly many *candidate keys* (specified using **UNIQUE**), one of which is chosen as the *primary key*.

o Any difference between these two tables ?

```
CREATE TABLE Enrolled1
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid,cid) )

CREATE TABLE Enrolled2
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid),
 UNIQUE (cid, grade) )
```

Primary and Candidate Keys in SQL

o Enrolled1: "For a given student and course, there is a single grade." vs. Enrolled 2: "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

o Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid,cid) )

CREATE TABLE Enrolled
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid),
 UNIQUE (cid, grade) )
```

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 ?

❖ E.g. *sid* is a foreign key referring to *Students*:

- Enrolled(*sid*: string, *cid*: string, *grade*: string)
- If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

Foreign Keys in SQL

❖ Only students listed in the *Students* relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Jazz101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- ❖ Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- ❖ What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- ❖ What should be done if a Students tuple is deleted?
 - Also delete all Enrolled tuples that refer to it.
 - Disallow deletion of a Students tuple that is referred to.
 - Set *sid* in Enrolled tuples that refer to it to a *default sid*.
 - (In SQL, also: Set *sid* in Enrolled tuples that refer to it to a special value *null*, denoting 'unknown' or 'inapplicable'.)
- ❖ Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

- ❖ SQL/92 and SQL:1999 support all 4 options on deletes and updates.
 - Default is **NO ACTION** (*delete/update is rejected*)
 - **CASCADE** (also delete all tuples that refer to deleted tuple)
 - **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
REFERENCES Students
ON DELETE CASCADE
ON UPDATE SET DEFAULT )
```

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.

How to design a good schema ?

- ❖ What do we mean by a "good design" ?
 - Can we quantify it ?
- ❖ Shall revisit this problem after finishing SQL
 - Theory of Normalization and Normal forms

The SQL Query Language

- ❖ The most widely used relational query language. Current standard is SQL-92.
- ❖ To find all 18 year old students, we can write:

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:
SELECT S.name, S.login

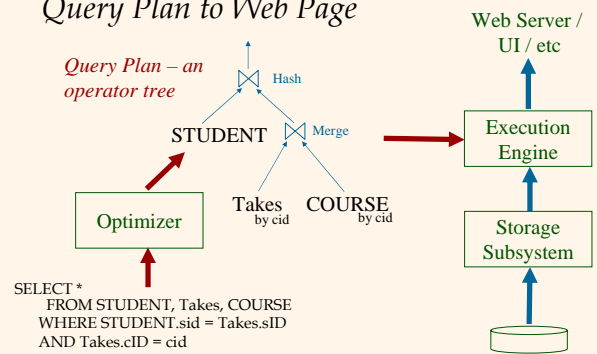
Codd's Relational Algebra

- ❖ A set of mathematical operators that compose, modify, and combine tuples within different relations
- ❖ Relational algebra operations operate on relations and produce relations ("closure")
 - f: Relation \rightarrow Relation
 - f: Relation \times Relation \rightarrow Relation

Codd's Logical Operations: The Relational Algebra

- ❖ Six basic (set) operations:
 - Projection $\pi_{\alpha}(R)$
 - Selection $\sigma_{\theta}(R)$
 - Union $R_1 \cup R_2$
 - Difference $R_1 - R_2$
 - Product $R_1 \times R_2$
 - (Rename) $\rho_{\alpha \rightarrow \beta}(R)$
- ❖ And some other useful ones:
 - Join $R_1 \bowtie_{\theta} R_2$
 - Semijoin $R_1 \ltimes_{\theta} R_2$
 - Intersection $R_1 \cap R_2$
 - Division $R_1 \div R_2$

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



Hint of things to come: Optimization Is Based on Algebraic Equivalences

- ❖ Relational algebra has laws of commutativity, associativity, etc. that imply certain expressions are equivalent in semantics
- ❖ They may be different in cost of evaluation!

$$\sigma_{c \wedge d}(R) = \sigma_c(R) \cup \sigma_d(R)$$

$$\sigma_c(R_1 \times R_2) = R_1 \bowtie_c R_2$$

$$\sigma_{c \wedge d}(R) = \sigma_c(\sigma_d(R))$$

- ❖ Query optimization finds the most efficient representation to evaluate (or one that's not bad)

An Equivalent, But Very Different, Formalism

- ❖ Codd invented a **relational calculus** that he proved was equivalent in expressiveness
 - Based on a subset of first-order logic - declarative, without an implicit order of evaluation
 - More convenient for describing certain things, and for certain kinds of manipulations

Summary - Relational Model: Advantages

- ❖ Data driven not design driven
 - designed once; data changes over time without affecting applications
- ❖ data stored, read, modified from "one" location
- ❖ rules/constraints control how data defined and enforced
- ❖ changes to database scheme without affecting application?

Example

- ❖ Storing information about Customers
 - One table
 - Cust(Last-name, First-name, SSN, address, telephone number)
- ❖ Suppose we want to add other types of telephones - such as mobile etc.??
 - Change schema ?

A better design – exploiting relational model

- ❖ Cust(Last-name, First-name, SSN, address, telephone number)
- ❖ Three table schema:
 - Cust (Last-name, First-name, SSN, Address)
 - Cust_Phone(SSN, number, phone_type)
 - Type_Detail(Type, Description)
 - example: (o, office), (h,house), (c, cell)

Next: Formal Query Languages

- ❖ Formal query languages
 - We focus mainly on relational algebra
 - This is what we will need when we discuss query optimization
 - Will glance at relational calculus
- ❖ Procedural vs Non-procedural languages
 - Can have a mix in practice