Introduction to Data Modeling

- A subset of the overall software design process, typically needed when the software being designed involves (duh) a back-end database management system
- Similar, but not identical, to generalized class or object modeling
- Main goals: **plan** and **communicate** your design
- Partial reference (primarily for entity-relationship modeling): SKS Chapter 6

- Other elements of software design are relevant to effective data modeling — they're all related:
  - *Learning the domain*: Problems to solve, tasks to perform, user needs, information needs, resource requirements (performance, storage) — whatever it takes to become an expert in the area that your software aims to address
  - *Defining use cases*: Use cases are individual episodes that define a task that your system performs; the overall set of use cases is called a use case model
- Never lose sight of your ultimate goal, which is to **plan** and **communicate** your design; don't fall into the trap of "documentation for documentation's sake"

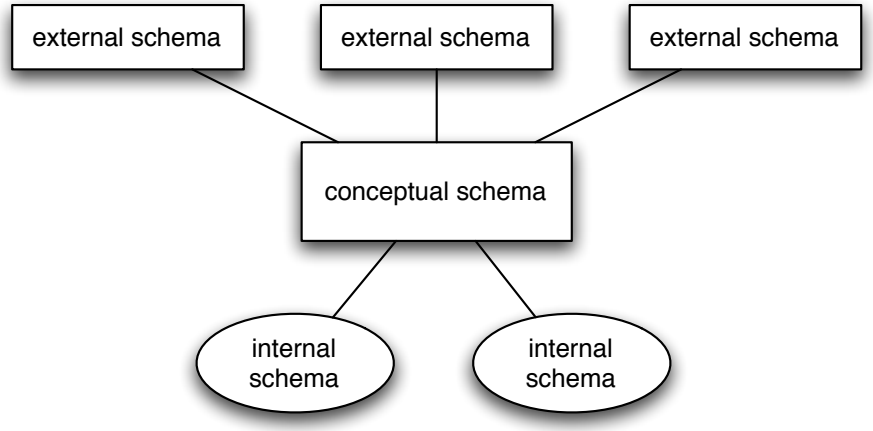
Database Design Phases

- *Characterize data needs* — part of learning the domain; results in *views* of the system
- *Conceptual design* — capture the entities/objects, relationships, and attributes of the domain/system
- *Functional requirements* — operations to perform; informed by the use case model, but more specific to database interaction
- *Database implementation* — translate the conceptual design into the chosen generalized database management system

- A conceptual database design doesn't yet care about the specific database management system (DBMS) that you've chosen for your software — nor should it
- Eventually, the rubber does need to meet the road, and so you need to implement the database in the system of your choice (such as PostgreSQL):
  - *Logical design* — translate or "map" your conceptual design in terms of your DBMS's chosen model: 90% of the time, this chosen model is the *relational data model*, which we will discuss later in the semester
  - *Physical design* — determine "physical" features of the database: typically, at this point, you're *extremely* performance and optimization oriented

The ANSI/SPARC Specification

Defined in 1977 as the layered approach for designing databases — similar concepts, some different terms:



Concept	ANSI/SPARC Term	SKS Term	Example(s)
Context-sensitive perception of the system (e.g. users)	external schema	view level	use case models, SQL views
Overall, master design of the system	conceptual schema	conceptual design	E-R model, UML model
Concrete implementation of the conceptual schema	internal schema	Two levels: logical design physical design	*Logical*: relational (most of the time), object-oriented *Physical*: DBMS-specific optimizations (paging, indexing, queries)

Most of the time, "data modeling" refers to the *conceptual level, schema, or design* — and therefore involves the E-R or UML modeling languages

Conceptual Data Modeling Languages

- Remember, the idea is effective **planning** and **communication**
- When you say *communication*, you need a *language*
- In the realm of databases, the *Entity-Relationship (E-R) Model* has reigned supreme for decades
- Lately, the *Unified Modeling Language* has taken hold, since it captures all of E-R's concepts and adds many new ones, such as object-orientation, use case models, activity/workflow, and system architecture

Key Modeling Concepts

Regardless of the language, a data model must capture the following information:

- *Entities* or *objects*: the "nouns" in your system
- *Relationships*: how entities are associated — the "verbs" in your system (e.g. Customer *buys* Products)
- *Attributes*: what information describes your entities and relationships? — these are your "adjectives"
- *Constraints*: what rules govern your system? — this is how the "real world" affects your system