Introduction to Data Management

Lecture #3
(Conceptual DB Design)

Instructor: Chen Li
Announcements

- HW #1 is now available
- Today’s plan – Conceptual DB design, *cont.*
  - Advanced ER concepts
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.
  - Dependent identifier is unique only within owner context (-----), so its fully qualified key here is (ssn, dname)

```mermaid
graph TD
  Employee[ssn] --> name[lot]
  Employee[ssn] --> premium
  Employee[ssn] --> Policy[1]
  Policy[1] --> N
  Policy[1] --> Dependents[dname, age]
```
Ternary Relationships (and beyond)

- A prescription is a 3-way relationship between a patient, a doctor, and a drug; with the cardinality constraints above:
  - A given patient+drug will be associated with one doctor (1)
  - A given patient+doctor may be associated with several drugs (N)
  - A given doctor+drug may be associated with several patients (M)

- (General) note: Relationship key ≤ (entity keys)
ISA ("is a") Hierarchies

- As in C++ or other PLs, ER attributes are inherited (including the key attribute).
- If we declare A ISA B, every A entity is also considered to be a B entity.

  - **Overlap constraints:** Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (Allowed or disallowed)
    - Ex: Hourly_Emps OVERLAPS Contract_Emps (else pick 1 of the 3 types)
  - **Covering constraints:** Does every Employees entity also have to be either an Hourly_Emps or a Contract_Emps entity? (Yes or no)
    - Ex: Hourly_Emps AND Contract_Emps COVER Employees (pick 1 of 2 vs. 1 of 3)

- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify subclasses that participate in a relationship.
- Design: specialization (top-down), generalization (bottom-up)
Aggregation

- Used when we have to model a relationship involving (entity sets and) a relationship set.
  - Aggregation allows us to treat a relationship set as an entity set for purposes of participating in (other) relationships.

Aggregation vs. ternary relationship:
- Monitors is a distinct relationship; even has its own attribute here.
- Each sponsorship can monitored by zero or more employees (as above).
Additional Advanced ER Features

- Multi-valued (vs. single-valued) attributes

- Derived (vs. base/stored) attributes

- Composite (vs. atomic) attributes

NOTE: Can model (two of) these using additional entity and relationship types.
Conceptual Design Using the ER Model

- **Design choices:**
  - Should a given concept be modeled as an entity or an attribute?
  - Should a given concept be modeled as an entity or a relationship?
  - Characterizing relationships: Binary or ternary? Aggregation? …

- **Constraints in the ER Model:**
  - A lot of data semantics can (and should) be captured.
  - But, not all constraints cannot be captured by ER diagrams. (*Ex:* Department heads from earlier…!)
Entity vs. Attribute

- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends how we want to use address information, the data semantics, and also the model features:
  - If we have several addresses per employee, *address* must be an entity if we stick to basic E-R concepts (as attributes cannot be set-valued w/o advanced modeling goodies).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic w/o advanced modeling goodies).
  - If the address itself is logically separate (e.g., the property that’s located there) and refer-able, it’s **rightly** an entity in any case!
Works_In4 does not allow an employee to work in a department for two or more periods. (Q: Why...?)

Similar to the problem of wanting to record several addresses for an employee: We want to record several values of the descriptive attributes for each instance of this relationship. Could be accomplished by introducing a new entity set, Period.
Entity vs. Relationship

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.
- What if a manager gets a discretionary budget that covers all managed depts?
  - Redundancy: \textit{dbudget} stored for each dept managed by manager.
  - Misleading: Suggests \textit{dbudget} associated with department-mgr combination.

\begin{itemize}
  \item Also note ISA and the relationship...
\end{itemize}
If each policy is owned by just 1 employee, with each dependent tied to their covering policy, first diagram is inaccurate.

Q: What are the additional constraints in the 2nd diagram? (And what else was wrong with the 1st diagram? 😊)
Binary vs. Ternary Relationships (Cont’d.)

- Previous example illustrated a case when two binary relationships were better than one ternary relationship.
- An example in the other direction: a ternary relation **Contracts** relates entity sets **Parts**, **Departments** and **Suppliers**, and has descriptive attribute **qty**.
Binary vs. Ternary Relationships (Cont’d.)

Bad design:

- S “can-supply” P, D “needs” P, and D “deals-with” S does not imply that D has agreed to buy P from S.
- And also, how we record qty?

![Entity-Relationship Diagram](image-url)
An example in the other direction: a ternary relation **Contracts** relates entity sets **Parts**, **Departments** and **Suppliers**, and has descriptive attribute *qty*:
Database Design Process (Flow)

- Requirements Gathering (interviews)
- Conceptual Design (using ER)
- Platform Choice (which DBMS?)
- Logical Design (for target data model)
- Physical Design (for target DBMS, workload)
- Implement (and test, of course 😊)

(Expect backtracking, iteration, and also incremental adjustments – and, we will actually be giving you a bit of practice with that last one in the next few HW assignments...! 😊)
Summary of Conceptual Design

- Conceptual design follows requirements analysis
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: entities, relationships, and attributes (of entities and relationships).
- Some additional constructs: weak entities, ISA hierarchies, and aggregation.
- Note: There are many variations on ER model (and many notations in use as well) – and also, UML...
Summary of ER (Contd.)

Several kinds of integrity constraints can be expressed in the ER model: cardinality constraints, participation constraints, and overlap/covering constraints for ISA hierarchies. Some foreign key constraints are also implicit in the definition of a relationship set (more about those will be coming soon).

- Some constraints (notably, functional dependencies) cannot be expressed in the ER model.
- Constraints play an important role in determining the best database design for an enterprise.
Summary of ER (Contd.)

- ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use an ISA hierarchy, and whether or not to use aggregation.

- Ensuring good database design → The resulting relational schema should be analyzed and refined further. FD information and normalization techniques are especially useful (coming soon).