**Content Models**

**ER Models (just the basics)**

**INLS 582, Systems Analysis**

**Monday, 3/19/2012 & Wednesday, 3/21/2012**

**Reading:**

[Elmasri, R. & Navathe,  S. (2007).  Ch. 3, Data modeling using the entity-relationship (ER) model, sections 3.1-3.4](http://libproxy.lib.unc.edu/login?url=http://eres.lib.unc.edu/coursepage.asp?cid=2001). In *Fundamentals of Database*, 5th edition. Addison Wesley.

**Assignments:** Wednesday 3/21/12, Project Draft Models1 & Progress Reports due.

Wednesday 3/21/12, Assign ER models, due Monday 4/2/12

**Slides:** ContentModels.pptx, ER\_Comp.pptx

Content models capture data, information "stuff"

Remember that content and process must coordinate and evolve together, otherwise risk of breakdowns.

What the inforamtion handles (content) and How the information system handles it (process)

Information life cycle: input, store, consult, transform, output; create and destroy

Consider:

semantics – what the information means in the context of the users, system, across related systems. Definitions, representation, behavior

Form and format must be supportive of use. e.g., structured, unstructured. Representation in artifacts.

related policies, administrative needs, constraints, standards, "customs" of use.

Examples of Content models

* ER (DB)
* Object models (DB, OO systems)
* CRUD (information life-cycle)
* Data dictionary (DB, authority control, definitions)
* Artifact
* Web site, web page architecture
* Card-sorting (analysis tool and model-creation)
* TOC, extended outlines (for documents)

Where is the content? DB, Spreadsheet, webpage/site, metadata, index, TOC, recordings, learning object, web logs, catalogue, training manual calendar/schedule, policy manual wite paper, report, exhibit, directory, data set, blog/wiki, email, posters, papers, budget, purchase orders, receipts

**ER Models and DB design.**

NOTE: the plan here is to go back and forth between ER and the familiar table, thinking about mapping and reverse engineering.

The ER model represents the objects (entities), attributes, and relationships in the system at a conceptual level. It also models generalizations and specializations. This is a static mode, which doesn't include any concept of processes, activities, or the order in which things occur.

Note on notation: There are lots of different styles of notation for ER models. We'll be using one variant here. Just remember that the essence of the model is the same, regardless of the notation, and be flexible!

ER model is conceptual model, not DD/implementation. A single ER may be implemented in a variety of ways, depending on design decisions (based on relational theory, characteristics of data, performance needs).

Represent information the DB must record for some period of time. (seconds to forever). The "footprint" of a transaction, record of real-world event. Entities and relationships between them.

Data elements need to represent the entities. An entity is a "bundle" of data elements.

Select the data type appropriate for each data element, define the behavior of each entity and data element.

**Feline Retirement Home.**

First table: representing **animals**. The structure of the table (title, rows, columns) tell us about what it represents.

The table is a familiar artifact, content container. But assumptions have already been made about what is in it. The ER is a higher-level, conceptual model for design, reverse engineering.

**Entity and Attributes in the ER**

**Entity:** object, thing, box, animal

An entity is a "thing" in the real world. It could be a physical object, an event or a conceptual object. Each entity has a name. The instances, or individual occurrences of the entity, have unique identifiers. In a DB, this is the primary key. This allows the instance to be referenced unambiguously. What you are doing is defining a class, or type of entity. In the real world, you will have one or more instances of this type, or members of this class. In an ER diagram, the entities are represented by labeled boxes.

**Attributes**

Each entity has attributes. These are the "facts" that we know about it and want to record in some way. The combination of these attributes are the representation of the entity to the information system. In the ER diagram "definitions', you determine what attributes the entity has. Instances of the entity have actual values for the attribute.

An attribute may have one or more values. These are generally modeled as attribute-value pairs.

A **single-valued attribute** is one where each entity will have at most one value for it. E.g., *SS#.*

A **multi-valued attribute** is one where an entity may have more than one value for it at the same time. E.g., *computer account name, telephone#.*

A **stored attribute** is one that is defined in the database and is permanently stored. In contrast, A **derived attribute** is one whose value can be determined or calculated from one or more other attributes. E.g., *age* is derivable from *birthday* and *current date*. Calculated field. Calculated as part of query or report.

A **default** value is a value that is likely to be used for most records/instances. Good example: NC as state of in students' current address. Bad example: name.

Primary Key: In relational DB design, each entity must have a unique identifier, or primary key. This is an attribute or combination of attributes that will have a unique value for each instance. This attribute can be modeled in the ER diagram.

FRH: ER and table: mapping and RE.

second table: enclosure. Given description, draw the ER and construct the corresponding table structure.

Defining Attributes:

Datatype: text, integer, floating point, date, currency, boolean, URL

Domain, legal, possible values, ranges.

Behavior : unique valur, PK, required/optional; single/multi, derived.

FRH: relationship between animal and enclosure. <slide>

Foreign Key in Animal represents the LivesIn relationship: where an animal lives.

**Relationships**

associations between entities.

Degree, Cardinality, Participation.

A relationship is an association between entities. Like entities, relationships also come in **types** and **instances**, where the type is the definition, and the instance is an actual occurrence of a relationship between entity instances

The relationship type defines the relationship in terms of the **participating entity types**, those entities between which the relationship exists. For example, the *car-owner* relationship has 2 participating entities; adults, and cars.

The relationship type defines the relationship in terms of the participating entity types, those entities between which the relationship exists. For example, the car-owner relationship has 2 participating entities; adults and cars. In the banking example, we have the relationship representing a customer taking out a loan, with 2 participating entities, customer and loan.

A binary relationship is one with 2 participating entity types. A ternary relationship has 3 participating entity types. It is very rare to see more. In fact, some designers recommend decomposing a ternary relationship to binary ones, when it can be done without losing information.

The **cardinality** of a relationship specifies how many relationship instances any entity instance can participate in. These are "rules" that describe how things work in the world, according to natural laws, legal constraints, business guidelines, etc. These are generally defined using binary relationship terminology, extend them for relationships of higher degree.

**One-to-one, 1:1**, means that each entity can only participate once. For instance, a manager can only manage one department, and each department can only have one manager.

**One-to-many, 1:M**, means that one entity can participate once, the other can participate many times. For example, a manager may manage many employees; each employee has only one manager. Many employees have the same manager.

**Many-to-many, M:N**, means that every entity can participate many times. For instance, an employee may work on several projects, and each project can have several employees. Or students and classes.

**Cardinality**

The **cardinality** of a relationship specifies how many relationship instances any entity instance can participate in. These are "rules" that describe how things work in the world, according to natural laws, legal constraints, business guidelines, etc. These are generally defined using binary relationship terminology, extend them for relationships of higher degree.

**One-to-one, 1:1**, means that each entity can only participate once. For instance, a manager can only manage one department, and each department can only have one manager.

Or, a proposal describes one project: each additional project must have its own proposal, and a proposal can describe only 1 project.

**One-to-many, 1:M**, means that one entity can participate once, the other can participate many times. For example, a manager may manage many employees; each employee has only one manager. Many employees have the same manager.

**Many-to-many, M:N**, means that every entity can participate many times. For instance, an employee may work on several projects, and each project can have several employees. Or students and classes.

A customer can have many savings accounts, and a savings account may have more than 1 owner. Similarly, a customer may take out many loans, and a loan may have more than one borrower. Consider: might a bank want to put some kind of limit on the number of loans a customer has? or the dollar amount of loans? How might we do this? Where would it be enforced?

These are constraints on how entities are expected to behave in the world; e.g. it is expected that a manager should manage one department, if there is a situation in which this doesn't seem to be the case, either the world is represented wrong in the model, or there is an error (e.g., the fact that a manager has transferred from one department to another was not correctly recorded.) These constraints can also be used to enforce rules. For instance, a class is supposed to be involved in a M:N relationship - if there is only one student enrolled, the class should be canceled.

**Participation**

The participation constraints on a relationship indicate whether each entity instance **must** participate in a relationship instance in order to exist. This comes in two types.

**Total participation** means that each entity **must** participate. For example, a savings account must be owned by at least on customer, no "free-floating" accounts.

**Partial participation** (or optional participation) means that only a subset of the entity instances are expected to participate; the instance can still exist without participating. For example, not every employee manages other employees.

The participation constraint is also a constraint on the actions in the real world - e.g. policy decisions. Consider students and teachers. Is a student a student in the term that he or she is not registered for classes? In other words, must each student totally participate in the *registered-for-class* relationship? Is a teacher a teacher in the term that he or she is not teaching a class?

**Relationship attributes**

In some sense, we can think of a relationship as being an intangible "thing" in the world too; and it can have its own attributes. When this occurs, you are reflecting the situation in the world where there are facts that are not controlled only by one entity, but rather by their combination in a relationship. This is highly context-sensitive, and will depend on the particular application.

For example, suppose you have students and classes, and the M:N relationship *register-for*. Students register for classes. How are you going to model the grade the student receives for the course? It is certainly associated with the student, but one student probably has many grades, each associated with a particular course. It is also associated with the course, but each course has many grades, each associated with a particular student. The grade can be viewed as being an attribute of the association of a particular student with a particular course; an attribute of the relationship.

Another example: customer orders item. How many of the item?

**Mapping between ER diagrams and relational model.** There are some basic steps that make this a fairly straightforward process.

1. For each entity, create a relation (table), with all its attributes in their simple (non-composite) form. Identify a primary key (1 or more attributes).

3. 1:1 binary relationships will be modeled as foreign keys in one of the participating entity tables. So the primary key of one entity will be one of the attributes in the other entity's table. In addition, any attributes of the relation will also be an attribute in that table. If one of the entities has total participation in the relationship, that entity should be the "home table" of the relationship.

Note: If both entities have total participation, and don't participate in any other relationship, you could merge the entities into one table.

4. 1:N binary relationships are also modeled using a foreign key. The primary key of the entity at the 1 side of the relationship is included as a foreign key in the table of the other entity. Also include any attributes of the relationship.

5. M:N binary relationships are modeled in a separate relation. This is because each instance in one entity is related to possibly many in the other, and vice versa. Since we can't have repeating fields in the relational model, this information must be extracted. The new table will contain the primary keys of each participating entity, along with any attributes of the relationship. The primary key of the table will be the combined primary keys of the participating entity.

<discuss options for relational integrity enforcement for each>

6. Multi-valued attributes must also be pulled out to form their own table. Again, this is because we can't have repeating fields. This table will include the attribute and the primary key (as foreign key) of the entity. The primary key will be the foreign key and the attribute.

**Mapping for N-ary relationships greater than binary**.

Similar to a M:N relationship. Create a new table containing the primary keys from each of the participating entities, along with any attributes of the relationship. The PK of the table will be the combined PKs, or you may create an autonum ID.

Next Step: M:N relationship with attribute.

FRH (4) the Staff develop Enrichment activities for the animals. These can be entertaining, training/exercise, help medical problems, and so on.

Need: title, purpose, description, target age. N:N relationship with date between animal and activity.

Draw ER with correct mapping.

FRH (5) Animals have favorite spots. Add multi-valued attribute to animal.

Show full schema (another content model) with FK arrows.