Introduction to Databases, Fall 2005

IT University of Copenhagen

Lecture 3: E/R modeling

September 12, 2005

Lecturer: Rasmus Pagh

Today's lecture

- What you should remember from last week.
- Data modeling - why and how?
- The basic entity-relationship (E/R) data model.
- E/R concepts and examples.
- Rules for designing a good E/R model.
- How to go from an E/R data model to a relational database schema.
- Advanced E/R concepts and examples.

What you should remember from last week

In this lecture I will assume that you remember:

- The concepts of the relational data model:
  - Data model
  - Relation schema
  - Database schema (i.e., the collection of relation schemas)
  - Attributes
  - Relation instance

What is data modeling?

Data modeling is the process of creating a data model for a given set of data.

Data modeling is a major part in developing software (no matter what programming language is used).

For programming relational databases, the data modeling should ultimately produce a relational data model.

Why data modeling?

The way data is stored is *very important* for the way in which it can be accessed and manipulated using SQL.

A good data model is one for which:

- It is easy to write correct and understandable queries.
- No change is needed for minor changes in the problem domain.
- If the problem domain changes significantly, it is easy to modify the data model and associated programs.

Sometimes it is also necessary to consider the data model's impact on the efficiency of database operations.

Database design methodology

Today and next week we will cover the dominant design methodology for relational databases, which consists of three steps:

1. Identify all relevant **E**ntities and **R**elationships, and describe them using so-called **E/R diagrams**.
2. Convert the E/R model to a number of relation schemas.
3. Eliminate (or reduce) redundancy by splitting relations. This process is called **normalization**. (Next week.)

We will focus explicitly on the aspects related most closely to relational databases, and not talk about other systems design methods such as formulation of business rules.

Next: The basic entity-relationship (E/R) data model

What is an entity?

An **entity** is an object of some sort (not necessarily physical).

Examples of entities:

"Die Hard" (an abstract object), "Bruce Willis" (a physical object).

We group similar entities into **entity sets**.

Examples of entity sets: "Movies", "stars".

Attributes of entity sets

To every entity set we associate a number of **attributes**, which can be thought of as properties of the entity set.

Example: The entity set "stars" might be given attributes *name*, *address*, *gender*, and *birthdate*.

Values of attributes:

- For each entity in the entity set, the attributes will be associated with values (e.g., the name, address, gender and birthdate of Bruce Willis).
- Values must be **atomic**, which essentially means they cannot be sets.

Example: An attribute *addresses* supposed to hold several addresses is not atomic, and hence not allowed.

Relationships between entity sets

Two entity sets may be connected in some way. In the E/R model this can be expressed as a **relationship**.

Example:

The "movies" and "stars" entity sets are connected by actors starring in movies. For example, "Die Hard" is connected to "Bruce Willis".

Formally, a relationship between entity sets A and B is a set of tuples (a,b), where entity a belongs to A, and b belongs to entity B.

Didn't we see this definition before? *Answer in two slides. . .*

How to identify entities

In *most* cases, an entity can be identified **uniquely** within its entity set by the values of its attributes.

> A set of attributes whose values uniquely identify an entity in its entity set is called a **key** of the entity set.

I.e., there can be no two entities with the same values on the key attributes.

Example:
The CPR-number attribute is a key for the entity set of Danish citizens.

If there are several possible keys, we choose one of them as the **primary key**. (If you already worked with SQL before the course, you may know that concept also from there.)

Relationships are relations

What is called a relationship in the E/R model is *exactly* what we defined last time as a relation.

- Entity sets correspond to the domains of the values.
- Tuples correspond to connections between entities.
- We can draw relationship instances as a table, like relation instances:

Movies	*Stars*
Basic Instinct	Sharon Stone
Total Recall	Arnold Schwarzenegger
Total Recall	Sharon Stone

Note: We use the attribute values of a key for each entity set to identify entities (unrealistically assuming that name is a key for Stars and Movies).

E/R diagrams

E/R diagrams are graphical notation for E/R data models. In this course we use the notation described in GUW (there are many variants of E/R notation).

Basics:

- Attributes are represented by ovals.
- Relationships are represented by diamonds.
- Entity sets are represented by rectangles, with lines to its attributes and relationships.

If an entity set has one or more keys, we underline the attributes in its primary key.

`[Figure 2.17 shown on slide]`

Problem session (5 minutes)

Suppose we want to design a database for a bank, with information on customers and their accounts, including:

- Names
- Addresses
- Phone numbers
- Customer type (e.g. business or private)
- CPR numbers
- Account numbers
- Account types
- Account balances

Design an E/R diagram for the database.

Next: E/R concepts and examples

Multi-way relationships

Relationships may connect *any* number of entity sets.

For example, a relationship among entity sets A, B, and C is a set of triples (a,b,c), where a, b, and c come from A, B, and C, respectively. Intuitively, the entities in these triples "are connected" or "belong together".

`[Figure 2.4 from Example 2.5 shown on slide]`

Note: This is different from having three relationships among A-B, B-C, and A-C. (Why?)

Multiplicity of relationships

We use arrows to indicate the **multiplicity** of a relationship.

An arrow from a relationship R to an entity set E means that:

> Every set of entities in the other entity sets of R is connected by R to **at most one** entity in E.

`[Figure 2.3 from Example 2.4 shown on slide]`

The multiplicity of a **binary** relationship (i.e., involving two entity sets) can be categorized as **one-one**, **many-one**, or **many-many**.

Attributes on relationships

Sometimes it is convenient to associate attributes with a relationship.

The meaning is that every tuple of entities connected by the relationship is associated with a value for each attribute.

In E/R diagrams, the attributes are drawn by connecting the relationship with an oval containing the attribute name.

[Figure 2.7 shown on slide]

Parallel and self-relationships

It is perfectly possible that there are several (different) relationships between the same entity sets.

[Figure 2.9 shown on slide]

Also, an entity set may be involved several times in the same relationship. In this case we label the lines with different name (called **roles**).

[Figure 2.5 from Example 2.6 shown on slide]

[Figure 2.6 from Example 2.7 shown on slide]

Next: From an E/R model to a relational database schema

From an entity set to a relation

We first consider *entity sets that have keys*, i.e., where there can be no two entities with the same attributes.

The corresponding relation then simply has the attributes of the entity set as its attributes. Easy!

[Figure 2.17]

From a relationship to a relation

We assume that the entity sets connected by the relationship all have a key.

The relation corresponding to a relationship then has the following attributes:

- The primary key attributes for each entity set involved in the relationship.
- Any attributes of the relationship itself.

It may be necessary to rename attributes if there are duplicate names.

`[Figure 2.17, 2.12]`

Possibilities for combining relations

Sometimes the translation to relation schemas results in more relations than necessary (or convenient).

Example: If there is a *many-one relationship* R between entity sets E and F, we can combine the relations corresponding to E and R into a single relation with a schema consisting of:

- The attributes of E,
- the key attributes of F, and
- any attributes belonging to R.

[Figure 2.17]

Next: Advanced E/R concepts and examples

Modeling of constraints

In addition to *what data* is stored, it is often important for a data model to specify *properties* of the data stored.

Such properties are expressed as **constraints**:

"Not all values are possible, only those where. . ".

We already saw examples of **single value constraints** in the E/R model:

- Keys.
- Many-one relationships.

As we saw, such constraints are useful when converting the E/R model to a relational schema.

Other kinds of constraints

- *Referential integrity constraints*: Exactly one value exists in a given role.
- *Domain constraints* restricting the value of some attribute.
- *Multiplicity constraints* on the number of entities connected to a single entity by a relationship.
- *General constraints*, expressed in a constraint-expression language (more on this in the lecture covering chapter 7).

Constraints in E/R diagrams

In E/R diagrams we may:

- Indicate by a rounded arrow the referential integrity constraint that the entity connected by some relationship is required to exist.
  [Figure 2.18]
- Indicate multiplicity constraints on the lines connecting a relationship.
  [Figure 2.19]

Constraints in the database schema

Often, constraints are part of the database schema, and *checked* by the DBMS in connection with updates.

Updates that violate a schema constraint are either not allowed, or cause other data to change in order to satisfy the constraint. (Lecture 8.)

Example: In SQL, the schema

```
MovieStar(
name CHAR(30) PRIMARY KEY,
address CHAR(255),
gender CHAR(1),
birthday DATE)
```

declares the constraint that `name` is a key for `MovieStar`.

Weak entity sets

Sometimes entities in an entity set are not uniquely identified by their attribute values.

This may happen, e.g., when entities in one entity set are subunits of other entities. `[Figure 2.21]`

> But we need a key to do the conversion to relations..!

To get a key for a weak entity set E, we need to use attributes of other entity sets, namely the key attributes of the entities to which it has **supporting relationships**.

Supporting relationships, intuitively

Suppose each entity e in a weak entity set E is connected to exactly one entity f in another entity set F.

Then the attribute values of f can also be thought of as belonging to e.

Hence, the attributes of F may be used to help distinguish entities in E.

[Figure 2.21]

Supporting relationships, formally

Suppose that E is a weak entity set, and the F is an entity set connected to E by a relationship R.

R is a supporting relationship for E if:

- It is a binary, many-one relationship from E to F.
- We can put a rounded arrow on the line from R to F in the E/R diagram, i.e., there is a unique entity in F corresponding to every entity of E.

Notes:

- If F has several supporting relationships to E we need to include the key attributes of E several times.
- E may itself be a weak entity set, so its keys may come from other entity sets.

Weak entity sets in E/R diagrams

- A weak entity set is drawn as a rectangle with **double** border.
- A supporting relationship is drawn as a diamond with **double** border.
- We underline any attributes of a weak entity set that are part of its key.

[Figure 2.21]

Subclasses

Sometimes one entity set in an E/R diagram is included in another one, i.e., all entities of one are also entities of another.

Example:
The entity set "Employees" is included in the entity set "People". We refer to "Employees" as the **subclass** and to "People" as the **superclass**.

This could be modeled using a 1-1 relationship, but many E/R notations provide notation for this kind of relationship (in GUW a triangle labeled *isa*).

`[Figure 2.10]`

Note that it would introduce redundancy to repeat attributes of the superclass in the subclass.

Converting subclass structures to relations

The book discusses three possible approaches:

- *Follow the E/R viewpoint.* Treat the *isa* relationship as any other relationship. The relation of the subclass will have the key of the superclass as an attribute.
- *Entities as objects.* Create a relation for each possible "kind of entity", i.e., for all possible subsets of attributes. Note that the number of relations can explode in some cases.
- *Use null values.* One relation having all attributes. Attributes that do not apply to an entity are given the value `NULL`.

The first one is always a safe choice, but there may in some cases be performance advantages (and even ease of programming advantages) for the second and third.

Next: Final remarks

Design principles for data modeling

Good data modeling can be difficult, as there may be many choices of what data model to choose.

The following design principles can be useful when considering a design:

- Be faithful to the specification of the application.
- Avoid duplication and other redundant information.
- The KISS principle (Keep It Simple, Stupid). [Figure 2.11]
- Choose the right relationships. [Figure 2.17, 2.12]
- Use attributes when possible. [Figure 2.17]

Bad Example: [Figure 2.14]

Sanity checking your E/R diagram

Many bad or wrong E/R diagram designs can be detected by considering:

1. When I convert the diagram to relations, can the resulting relations accommodate the data I tried to model?
2. Have I used relations in all the places that I should? Entity sets should never have attributes that reference (the key of) another entity set.
3. Have I used attributes whose value is a set? (This can be specified in some E/R notations, but is not allowed in the one of GUW.)

E/R model vs UML

UML is an increasingly popular notation that can be used for object-oriented design and analysis, also for database systems.

Similar features:

- Entity type $\approx$ class.
- Entity instance $\approx$ object (instance).
- Relationship $\approx$ association.
- Super/subtypes $\approx$ inheritance

Some differences/additional features of UML:

- Implicit primary key in objects.
- Class-scope attributes.
- Notation for associating *actions/behavior* to objects.
- Notation for showing example object instances.

Most important points in this lecture

As a minimum, you should after this week:

- Be able to use the E/R data model to describe a database, including:
  - Choice of keys.
  - Deciding whether relationships are one-one, many-one, or many-many.
  - Deciding referential integrity constraints.
  - Drawing the E/R data model as an E/R diagram.
- Be able to convert an E/R data model to a relational database schema.

Next time

Next week we will see how to use so-called **normalization** to improve a relational database schema:

- Functional dependencies – the "bad guys" we wish to eliminate.
- Splitting relations to remove functional dependencies.
- Goal: Get the schema in "normal form" (3NF, BCNF, or 4NF).